ezlo innovation
mios vera fortrezz centralite

Table of contents

Plugins **6**
- Plugin Config File 6
- Flow 7

Cloud API **8**
- hub.extensions.plugin.run 8
- Examples 8

Lua API **9**
- Events subscription 9
- Error handling 10
- Modules 10
  - Core module 10
- Core module API (require "core") 11
- Getters 11
  - get_gateways() 11
  - get_gateway() 11
  - get_devices() 12
  - get_item() 14
  - get_items() 17
  - get_items_by_device_id() 19
  - get_current_temperature_scale() 22
- Adders 22
  - add_device() 22
  - add_item() 24
- Removers 26
  - remove_device() 26
  - remove_device_sync() 27
  - remove_item() 27
  - remove_gateway_devices() 28
  - remove_item_dictionary_value() 28
- Updaters 29
  - update_item_value() 29
  - update_item_value_with_min_max() 29
  - update_item_dictionary_value() 30

update_reachable_state() 30
update_ready_state() 31
modify_device() 31
modify_item() 34
Others 36
generate_item_dictionary_number_id() 36
send_ui_broadcast 37
Values 37
Scalable 38
RGB 38
User Code 38
Button State 39

Storage module 40
Storage module description 40
Storage module API (require "storage") 40
get_bool(key), get_number(key), get_string(key), get_table(key) 40
set_bool(key, value), set_number(key, value), set_string(key, value), set_table(key, value) 40
exists(key) 41
delete(key) 41
delete_all() 42

Network module 43
Network module description 43
How to use example: 43
Network module API (require "network") 45
Network events 45
Name of a network event: "network" 46
Structure of a network event: 46
Types of network events: 46
Structure of an io_activity event: 46
Type of an I/O activity: 47
Network constans: 47
subscribe() 47
connect() 48
send() 49

receive() 49
inspect_data() 50
inspect_data_size() 50
set_handler() 51
close() 51
Timer module 53
Timer module description 53
Timer module API (require "timer") 53
Functions 53
set_timeout 54
set_timeout_with_id 55
set_interval 56
set_interval_with_id 57
cancel 58
exists 59
reset 60

Zwave module **63**
Zwave module description 63
Zwave module API (require "zwave") 63
reset() 63
start_include() 63
start_exclude() 64
subscribe() 64
unsubscribe() 65
set_device_specific_key() 65
set_auth_mode() 65
stop_include() 66
stop_security_include() 66
get_all_nodes_ids() 67
get_node() 67
get_command_class() 68
set_value() 68
set_value_multicast() 69
set_token_value() 69
set_token_value_multicast() 69
set_rgb_value() 70

set_rgb_value_multicast() 71
set_sound_switch_configuration() 72
request_sound_switch_configuration() 72
set_sound_switch_tone_play() 73
request_sound_switch_tone_play() 73
set_door_lock_mode() 73
set_setpoint_value() 74
set_configuration_value() 74
reset_configuration_value() 75
switch_multilevel_level_change() 75
request_value() 76
request_node_values() 76
request_setpoint_value() 77
request_user_code() 77
request_configuration_value() 78
request_association_group() 78
add_node_to_association_group() 79
remove_node_from_association_group() 79
set_user_code() 80
set_silent_sensors() 80
request_alarm_sensor_value() 80
set_week_day_entry_lock_schedule 81
request_week_day_entry_lock_schedule 82
set_daily_repeating_entry_lock_schedule 82
request_daily_repeating_entry_lock_schedule 83
set_year_day_entry_lock_schedule 83
request_year_day_entry_lock_schedule 84

ZWave events **85**
Type of events: 85

ZWave Values **111**
User id statuses 111
User polling statuses 112
S2 authentication modes 112
Thermostat operating states 113
Door lock modes 114

Door condition 115
Latch states 115
Bolt states 115
Door states 115
Event statuses 116
Action types 116
Command 117
Alarm sensor types 118
Notification types 119
Color Switch Component IDs 121

Zwave lua objects 121
Node 121
Entry lock schedules 135
Week day entry lock schedule 135
Year day entry lock schedule 136
Daily repeating entry lock schedule 137
Command classes 138
Default 138
Associations 138
Association group info 139
Binary sensor 140
Central scene 140
Color Switch 141
Meter 141
Multi channel associations 142
Sound switch 143
Thermostat mode 144
Thermostat fan mode 144
Thermostat setpoint 145
User code 146
Schedule entry lock 147

Plugins

All plugins installed in the **/opt/firmware/plugins** folder on the controller.
Each plugin has its own subfolder which contains plugin config file and related scripts.

Install own plugin via SSH

Custom plugin can be installed via SSH

```
scp -r <PATH_TO_PLUGIN> root@<CONTROLLER_IP>:/opt/firmware/plugins
```

Firmware should be restarted:

```
service firmware restart
```

Plugin Config File

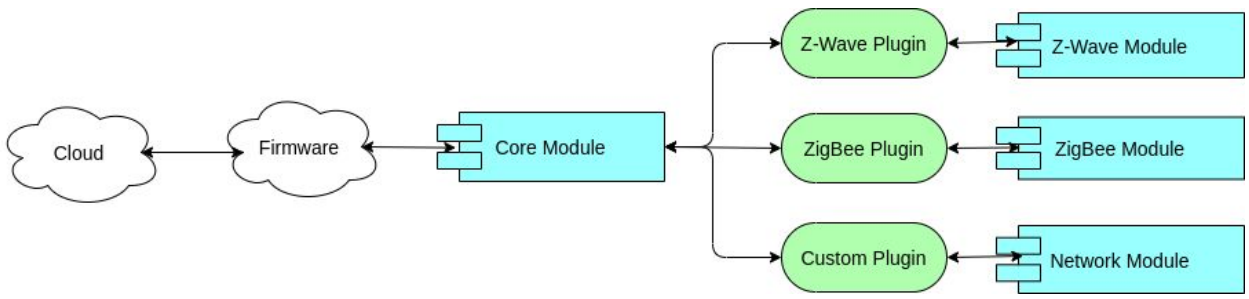
Each plugin must have own **config.json** file with next information:

Field	Description
`id`	unique id of plugin or unique name of plugin(used for dependencies resolving)
`version`	plugin version(used for dependencies resolving)
`type`	gateway - implementation of "smart" protocol extension - additional functionality plugin
`dependencies`	Required plugins and addons for this plugin
`permissions`	Script modules which needs for working this plugin
`startup`	script which will be run when all dependent modules will be started
`migration`	script which will be run before startup script. This script should run migration of plugin data
`teardown`	script which will be run before removing of plugin
`gateway`	gateway functionality description
`name`	Internal unique name of gateway

	`label`		Friendly gateway name for UI
	`reason`		required but not used
	`setItemValueCommand`		script for setting items value of this gateway
	`operations`		templates for specific operations(not used)
		`template`	path to template file(not used)
		`operation`	type of operation: adding, removing, device_settings, gateway_settings(not used)

Flow

Main role of plugin is converting protocol specific model to universal device abstraction model. Plugin must “understand” protocol specific models and behaviour and help firmware to work with it.



Cloud API

hub.extensions.plugin.run

Runs specified plugin script

request:

Field	Type	Required	Description
params.script	string	+	Plugin script id (same format as used inside plugin .lua scripts)

params.scriptParams	object	-	Parameters to be passed to the script

response:

<empty>

Broadcast	Description
hub.extensions.plugin.run.progress	Operation progress information

Examples

Execute start_include script of zwave plugin

request:

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

Lua API

The Hub provides Lua API to access functionality it implements. This functionality is splitted in groups, namespaces or modules, like zwave, extensions, core, etc. Such modules (as such provided functionality) can be used withing Lua scripts by using a standart mean - require.

Events subscription

Plugin can be subscribed for events (zwave, core, network, ...) using subsribe(scipt_name) method. When some event happens, a subscribed script gets called with an appropriate event (or, which is the same, plugin gets notified with this event).

Structure of a generic event:

```
{
      "event": "string",
      "data": {}
    }
```

fields	type	description
event	string	type of an event
data	table	data specific to the event

Error handling

In a case of errors methods throw exceptions, if otherwise not stated. An uncaught exception causes a script to be aborted, if it is not what you want, caught it by using pcall.

example:

```
require "network"

--network.connect( {} )  would cause the script to be aborted
op_res, result = pcall( network.connect, {} )

if not op_res then
  print( "fail to connect by some local reason: " .. result )
else
  print( "wait till a remote side accepts our request for connection, conn_hndl: " ..
result )
end
```

Modules

- core (API to an abstract model of gateways/devices/items)
- storage (API to a storage holding data between scripts launching)
- network (API to a network functionality (Berkeley sockets, REST requests))
- zwave (API to a ZWave functionality backed by a Zwave hardware)

Core module

Implements and supports the abstract model of gateways/devices/items to provide a universal way to control smart devices.

Gateway represents a communication domain or communication technology.

Device represents a smart device to be controlled/monitored, like smart outlet, smart lock, smart switch, etc.

Item represents a virtual control point [vcp], like turn on/off, set some value, get current value, etc. Such virtual control point [pcp] gets mapped to a physical control point like electrical relay, shift register, AC/DC meter, etc.

```
gateway: zwave
  device: smart_lock_1
    items: lock/unlock, set_password
```

Core module API (require "core")

Getters

get_gateways()

Get all registered gateways within the hub.

call: core.get_gateways()

params: none.

return: an array of gateways (table) or nill in a case of an error. Look at **get_gateway()** to get a gateway table format.

example:

```
  require "core"

   local gateways = core.get_gateways()
   if gateways then
     for _, gate in ipairs( gateways ) do
       print( "gateway name:" .. gate.name )
     end
   end
```

get_gateway()

Get a gateway registred for a plugin the current script belongs to.
call: core.get_gateway()

params: none.

return: a gateway (table) or nill in a case of an error.

fields	type	description
_id	string	an id of the gateway
name	string	a name obtained from a plugin's config
pluginId	string	an id of a plugin this gateway is a part of
label	string	a public name of the gateway
reason	string	a public reason of not 'ready' status

unreachableReasons	{ { ["some_key"] = string, ["some_key"] = string }, {...} }
unreachableActions	{ { ["some_key"] = string, ["some_key"] = string }, {...} }

example:

```
require "core"

local gateway = core.get_gateway()
if gateway == nil then
  print("fail to get a gateway")
end
```

get_devices()

Get all devices registred within the hub.

call: core.get_devices()

params: none.

return: an array of devices (table) or nill in a case of an error.

fields	type	description
_id	string	an id of the device
gatewayId	string	an id of a gateway this device belongs to
name	string	a public name of the device

category	string	a device category
subcategory	string	a device subcategory
type	string	
deviceTypeId	string	
roomId	string	an id of a room this device belongs to
security	string	[opt] Security level how the device is connected. Possible options: `no`, `low`, `middle`, `high`.
info	{ ["some_key"] = string, ["some_key"] = string }	some additional information for this device
batteryPowered	bool	whether the device is battery powered
reachable	bool	whether the device is reachable
persistent	bool	[opt] whether the device is persistent. False by default.

ready	bool	[opt] Ready status of device. `true` value means device is ready to any changes. `false` value means device is busy.

example:

```
  require "core"

   local devices = core.get_devices()
   if devices then
     for _, dev in ipairs( devices ) do
       print( "device name:" .. dev.name )
     end
   end
```

get_item()

Get item registered within the hub by specified id.

call: core.get_item()

params: item id (string).

return: item (table) or nill in a case of an error.

fields	type	description
_id	string	an id of the item
deviceId	string	an id of a device the item is being registering for
name	string	a type of the item

type	string [int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype]	a type of an item's value
hasGetter	bool	whether the item provides an ability to get a value
hasSetter	bool	whether the item provides an ability to set a value
show	bool	whether the item should be shown on UI
enum	table	set of values for value type 'token'
value	int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype	an item's value
valueMin	int : float : **scalable**	lower limit of item's value
valueMax	int : float : **scalable**	upper limit of item's value

elementsMaxNumber	int	max allowed elements of a dictionary or an array value
userCodeRestriction	string	restriction for a userCode code field
elementsMaxNumberPerArray	int	max allowed elements of each array (if array is subtype of dictionary or another array)
oneWeekDayCost	int	cost of one week day in `item weekly_user_code_intervals`
oneShiftedWeekDayCost	int	cost of one shifted (startTime > stopTime) week day in `item weekly_user_code_intervals`

example:

```
require "core"

local item = core.get_item( item_id )
if item then
    print( "item name:" .. item.name )
end
```

---

get_items()

Get all items registered within the hub.

call: core.get_items()

params: none.

return: an array of items (table) or nill in a case of an error.

fields	type	description
_id	string	an id of the item
deviceId	string	an id of a device the item is being registering for
name	string	a type of the item
type	string [int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype]	a type of an item's value
hasGetter	bool	whether the item provides an ability to get a value
hasSetter	bool	whether the item provides an ability to set a value
show	bool	whether the item should be shown on UI
enum	table	set of values for value type 'token'

value	int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype	an item's value
valueMin	int : float : **scalable**	lower limit of item's value
valueMax	int : float : **scalable**	upper limit of item's value
elementsMaxNumber	int	max allowed elements of a dictionary or an array value
userCodeRestriction	string	restriction for a userCode code field
elementsMaxNumberPerArray	int	max allowed elements of each array (if array is subtype of dictionary or another array)
oneWeekDayCost	int	cost of one week day in item `weekly_user_code_intervals`
oneShiftedWeekDayCost	int	cost of one shifted (start_time > stop_time) week day in `item weekly_user_code_intervals`

example:

```
require "core"
```

```
local items = core.get_items()
if items then
  for _, item in ipairs( items ) do
    print( "item name:" .. item.name )
  end
end
```

---

get_items_by_device_id()

Get all items registered within the hub related by specified device id.

call: core.get_items_by_device_id()

params: device id (string).

return: an array of items (table) or nill in a case of an error.

fields	type	description
_id	string	an id of the item
deviceId	string	an id of a device the item is being registering for
name	string	a type of the item
type	string [int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype]	a type of an item's value

hasGetter	bool	whether the item provides an ability to get a value
hasSetter	bool	whether the item provides an ability to set a value
show	bool	whether the item should be shown on UI
enum	table	set of values for value type 'token'
value	int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype	an item's value
valueMin	int : float : **scalable**	lower limit of item's value
valueMax	int : float : **scalable**	upper limit of item's value
elementsMaxNumber	int	max allowed elements of a dictionary or an array value
userCodeRestriction	string	restriction for a userCode code field

elementsMaxNumberPerArray	int	max allowed elements of each array (if array is subtype of dictionary or another array)
oneWeekDayCost	int	cost of one week day in item `weekly_user_code_intervals`
oneShiftedWeekDayCost	int	cost of one shifted (startTime > stopTime) week day in item `weekly_user_code_intervals`

example:

```
require "core"

 local items = core.get_items_by_device_id( device_id )
 if items then
   for _, item in ipairs( items ) do
     print( "item name:" .. item.name )
   end
 end
```

---

get_current_temperature_scale()

Get current temperature scale from configuration of hub.

call: core.get_current_temperature_scale()

params: none.

return: string ["celsius" | "fahrenheit"].

example:

```
require "core"
```

```
    scale = core.get_current_temperature_scale()
```

Adders

add_device()

Register a device for a specific gateway.

call: core.add_device()

params: device (table).

fields	type	description
gateway_id	string	[deprecated] an id of a gateway the device is being registering for
name	string	a name displayed by the UI
category	string	a device category
subcategory	string	a device subcategory
type	string	
device_type_id	string	
room_id	string	[opt] an id of a room this device belongs to

parent_device_id	string	[opt] id of parent device
info	{ ["some_key"] = string, ["some_key"] = string }	[opt] some additional information for this device
battery_powered	bool	whether the device is battery powered
reachable	bool	[opt] whether the device is reachable
persistent	bool	[opt] whether the device is persistent. False by default.
security	string	[opt] Security level how the device is connected. Possible options: `no`, `low`, `middle`, `high`. `no` by default.
ready	bool	[opt] Ready status of device. `true` value means device is ready to any changes. `false` value means device is busy.

return: device id (string) or nill if failed to add a device.

example:

```
  require "core"

   local gateway = core.get_gateway()
   local device = core.add_device( { gateway_id = gateway._id, ... } )
```

---

add_item()

Register an item for a specific device.

call: core.add_item()

params: item (table).

fields	type	description
device_id	string	an id of a device the item is being registering for
name	string	a type of the item
value_type	string [int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype]	a type of an item's value
has_getter	bool	whether the item provides an ability to get a value
has_setter	bool	whether the item provides an ability to set a value
show	bool	whether the item should be shown on UI

enum	table	[opt] set of values for value type 'token'
value	int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype	an item's value
value_min	int : float : **scalable**	[opt] lower limit of item's value
value_max	int : float : **scalable**	[opt] upper limit of item's value
elementsMaxNumber	int	max allowed elements of a dictionary or an array value
userCodeRestriction	string	restriction for a userCode code field
elementsMaxNumberPerArray	int	max allowed elements of each array (if array is subtype of dictionary or another array)
oneWeekDayCost	int	cost of one week day in item `weekly_user_code_intervals`

oneShiftedWeekDay Cost	int	cost of one shifted (startTime > stopTime) week day in item `weekly_user_code_intervals`

return: item id (string) or nill if failed to add an item.

example:

```
require "core"

local device = core.add_device( {...} )
local item = core.add_item( { device_id = device._id, ... } )
```

Removers

remove_device()

Remove a device specified by a device id and all items registered for this device.

call: core.remove_device()

params: device id (string).

return: none.

example:

```
require "core"

core.remove_device( device_id )
```

---

remove_device_sync()

Remove a device specified by a device id and all items registered for this device. Execution of this method will be finished after device deletion.

call: core.remove_device_sync()

params: device id (string).

return: none.

example:

```
require "core"

core.remove_device_sync( device_id )
```

---

remove_item()

Remove an item specified by an item id.

call: core.remove_item()

params: item id (string).

return: none.

example:

```
require "core"

core.remove_item( item_id )
```

---

remove_gateway_devices()

Remove all devices registered for a gateway specified by a gateway id

call: core.remove_gateway_devices()

params: gateway id (string).

return: none.

example:

```
require "core"

core.remove_gateway_devices( gateway_id )
```

---

remove_item_dictionary_value()

Remove value with specific key from specific dictionary item

call: core.remove_item_dictionary_value()

params: item_id (string), item_dictionary_id (string)

return: none.

example:

```
require "core"

item_id = "1234abcd"
item_dictionary_id = "4"

core.remove_item_dictionary_value( item_id, item_dictionary_id )
```

Updaters

update_item_value()

Update a specific value for a specified item (item is specified by id).

call: core.update_item_value()

params: item_id (string), value (it depends on a value type of a item).

return: none.

example:

```
require "core"

 value = true
 core.update_item_value( item_id, value )
```

---

update_item_value_with_min_max()

Update a specific value in specific range for a specified item (item is specified by id).

call: core.update_item_value_with_min_max()

params: item_id (string), value (it depends on a value type of a item), value_min, value_max.

return: none.

example:

```
require "core"

 item_id = "1234abcd"
 value = {
     value = 12,
     scale = "celsius"
 }
 value_min = {
     value = 0,
     scale = "fahrenheit"
 }
 value_max = {
     value = 34,
     scale = "celsius"
 }
 core.update_item_value_with_min_max( item_id, value, value_min, value_max )
```

---

update_item_dictionary_value()

Update value with specific key of specific dictionary item

call: core.update_item_dictionary_value()

params: item_id (string), item_dictionary_id (string), value (it depends of item dictionary subtype)

return: none.

example:

```
require "core"

item_id = "1234abcd"
item_dictionary_id = "4"
value = {
    userCode = "3141",
    userName = "Ivan"
}

core.update_item_dictionary_value( item_id, item_dictionary_id, value )
```

---

update_reachable_state()

Update reachable state for a specified device and all items registered for this device (device is specified by id).

call: core.update_reachable_state()

params: device id (string), reachable (bool).

return: none.

example:

```
require "core"

reachable = true
core.update_reachable_state( dev_id, reachable )
```

---

update_ready_state()

Update ready state for a specified device (device is specified by id).

call: core.update_ready_state()

params: device id (string), ready (bool).

return: none.

example:

```
   require "core"

    ready = true
    core.update_ready_state( dev_id, ready )
```

---

modify_device()

Modify existing device fields. (only for migration)

call: core.modify_device()

params: device_id, device (table).

fields	type	description
gateway_id	string	[deprecated] an id of a gateway the device is being registering for
name	string	a name displayed by the UI
category	string	a device category

subcategory	string	a device subcategory
type	string	
device_type_id	string	
room_id	string	[opt] an id of a room this device belongs to
parent_device_id	string	[opt] id of parent device
info	{ ["some_key"] = string, ["some_key"] = string }	[opt] some additional information for this device
battery_powered	bool	whether the device is battery powered
reachable	bool	[opt] whether the device is reachable
persistent	bool	[opt] whether the device is persistent. False by default.
security	string	[opt] Security level how the device is connected. Possible options: `no`, `low`, `middle`, `high`. `no` by default.

ready	bool	[opt] Ready status of device. `true` value means device is ready to any changes. `false` value means device is busy.

return: none.

Fields that will be modified
category
subcategory
type
security
name
parent_device_id

example:

```
   require "core"

    local gateway = core.get_gateway()
    local device = core.modify_device( device_id, { gateway_id = gateway._id, ... } )
```

---

modify_item()

Modify existing item fields. (only for migration)

call: core.modify_item()

params: item_id, item (table).

fields	type	description
device_id	string	an id of a device the item is being registering for
name	string	a type of the item
value_type	string [int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype]	a type of an item's value
has_getter	bool	whether the item provides an ability to get a value
has_setter	bool	whether the item provides an ability to set a value
show	bool	whether the item should be shown on UI

enum	table	[opt] set of values for value type 'token'
value	int : bool : float : string : **rgb** : **scalable** : **userCode** : **buttonState** : token : dictionary.#subtype : array.#subtype	an item's value
value_min	int : float : **scalable**	[opt] lower limit of item's value
value_max	int : float : **scalable**	[opt] upper limit of item's value
elementsMaxNumber	int	max allowed elements of a dictionary or an array value
userCodeRestriction	string	restriction for a userCode code field
elementsMaxNumberPerArray	int	max allowed elements of each array (if array is subtype of dictionary or another array)
oneWeekDayCost	int	cost of one week day in item `weekly_user_code_intervals`

oneShiftedWeekDay Cost	int	cost of one shifted (startTime > stopTime) week day in item `weekly_user_code_intervals`

return: none.

All item fields can be modified using this method.

example:

```
   require "core"

    local item = core.modify_item( item_id, { device_id = device_id, ... } )
```

Others

generate_item_dictionary_number_id()

Generates free number key for specific dictionary item in specific range

call: core.generate_item_dictionary_number_id()

params: item_id (string), min_id (int), max_id (int)

return: item_dictionary_id (int) or nil if no free ids in the range.

example:

```
   require "core"

    item_id = "1234abcd"

    item_dictionary_id = core.generate_item_dictionary_number_id(item_id, 1, 10)
    print(item_dictionary_id) -- output: 1

    core.update_item_dictionary_value( item_id, tostring(item_dictionary_id), "value1"
)
```

```
item_dictionary_id = core.generate_item_dictionary_number_id(item_id, 1, 10)
print(item_dictionary_id) -- output: 2
```

---

send_ui_broadcast

Send broadcast with custom data to ui.

call: core.send_ui_broadcast()

params: broadcast_data (table)

return: none.

Broadcast	Description
hub.extensions.plugin.ui_broadcast	Informs about some changes in plugins

example:

```
require "core"

core.send_ui_broadcast( { data = "data" } )
```

Values

Scalable

Scalable value

fields	type	description
scale	string	

value	float

example:

```
{
    "scale": "celsius",
    "value": 36
}
```

RGB

Rgb value

example:

```
{
    ...
    "red": 10,
    "green": 10,
    "blue": 10,
    ...
}
```

User Code

User code value

fields	type	description
code	string	
name	string	

example:

```
{
    "code": "some code",
```

```
        "name": "code name"
    }
```

Button State

Button state value

fields	type	description
button_number	int	
button_state	string	

example:

```
    {
        button_number = 1,
        button_state = "press_1_time"
    }
```

Storage module

Storage module description

Module provides an access to persistent storage. Plugins can store data in key/value format. Each plugin has its own volume in storage and implementation must guarantee no clashes between key's in different volume's As long as one plugin can use other ones and so on forming calling tree an implementation must use volume for the root plugin in this tree. It means that if plugin is meant to be used as library it will have separate storage for every topmost user. However, plugins in one calling tree should take responsibility to prevent key collision in one volume.

Storage module API (require "storage")

get_bool(key), get_number(key), get_string(key), get_table(key)

Get value from storage with corresponding key.

call: storage.get_***(key)

params: key - string

return: value if it exists and can be converted to specified type, nil otherwise

example

```
require "storage"
local val = storage.get_number("Pi")
```

set_bool(key, value), set_number(key, value), set_string(key, value), set_table(key, value)

Put value to storage with corresponding key

call: storage.set_***(key, value)

params: key - string, value - accordingly to method name

return: none

example

```
require "storage"
storage.set_number("Pi", 3.1415)
```

exists(key)

Check if key is set.

call: storage.exists(key)

params: key - string

return: true if key is set, false otherwise

example

```
require "storage"
if storage.exists("Pi") then
    print("Pi is stored")
else
    print("Pi is not stored")
end
```

delete(key)

Delete value from storage marked with key. Function has no effect if key does not exist.

call: storage.delete(key)

params: key - string

return: none

example

```
require "storage"
storage.delete("Pi")
```

delete_all()

Clear storage. Deletes all key/value pairs from current volume

call: storage.delete_all()

params: none

return: none

example

```
require "storage"
storage.delete_all()
```

Network module

Network module description

Module provides an access to a network functionality (async Berkeley sockets, REST requests).

How to use example:

```
--main.lua

--all methods throw exceptions in a case of errors, unless otherwise stated

params = ...

--[[
    params.ip (string) IPv4 / IPv6 in text representation or a host name
    params.port (string) port in text representation
    params.connection_type (string) [ "TCP" | "UDP" ]
    params.options (integer)
    params.data (string) binary data
--]]

net = require "network"

--subcribe for 'network' events (set a default events handler)
--[[todo: if we establish an event format we can allow plugin author
        to collect all events handling in one file if it needs it ]]
net.subscribe( "HUB:zwave/scripts/events_handling" )

--[[async request
    by default, io async notifications are sent to a script subscribed by
`network.subscribe()`, to change it use
    `network.set_handler()` --]]
hndl = net.connect( { ip = params.ip, port = params.port, connection_type =
params.connection_type },
                    params.options --[[ = 0 ]] )

if argv[ 1 ] == 1 then
    --an async flow

    --it's up to a plugin's author to make sure all data was sent
    local sent_amount = net.send( hndl, "data" )
```

```
    --`net.receive( hndl )` would return immediately with an empty result (empty
string)
elseif argv[ 1 ] == 2 then
    --an async flow, change the default events handler

    net.set_handler( hndl, "HUB:custom_events_hander" )
    net.send( hndl, "hello" )
else
    --an async flow of REST requests (to be done)

    local http = require( "network" ).http

    http.get( "http://example.com/customers", 5 )
    http.post( "http://example.com/customers/some_customer", { name = "Linux", age =
"27" }, 5, "root" )
end

net.close( hndl )

--event_hanlding.lua

params = ...

--[[
    {
      "event": "network",
      "data": {
        "event_type": "io_activity",
        "event": {
          "handle": 42,
          "io_activity_type": "IN"
        }
      }
    }
--]]

if params.event == "network" then
    local net = require "network"
    local ev_types = net.event_types
    local ev = params.data

    if ev.event_type == ev_types.io_activity.type_name then
        local ev = ev.event

        if ev.io_activity_type == ev_types.io_activity.CONNECTED then
            print( "a connection on hndl: " .. ev.handle .. " has been established" )
        end
```

```
        if ev.io_activity_type == ev_types.io_activity.FAILED_TO_CONNECT then
            print( "fail to establish a connection on hndl: " .. ev.handle )
        end
        if ev.io_activity_type == ev_types.io_activity.IN then

            --allows to make a decision depending on a size of received data
            if net.inspect_data_size( ev.handle ) < 10 then
                return
            end

            --allows to view data on a connection handler without fetching them
            local data = net.inspect_data( ev.handle )
            if #data > 20 and string.find( data, "OK" ) then

                --get up to 10 bytes, never blocks
                local data = net.receive( ev.handle, 10 )
                print( "hndl: " .. ev.handle .. " got a new data: " .. data )
            end
        end
        if ev.io_activity_type == ev_types.io_activity.OUT then
            net.send( ev.handle, "data" )
            print( "hndl: " .. ev.handle .. " some data has been sent" )
        end
        if ev.io_activity_type == ev_types.io_activity.CLOSED then
            net.close( ev.handle )
            print( "connection with hndl: " .. ev.handle .. " has been closed by a
remote side" )
        end
    end
end
```

Network module API (require "network")

network module supports only async mode.
Implementation keeps an internal buffer for each connection handler, which can be fetched by network.receive or inspected by network.inspect_data. Size of such buffer can be inspected by network.inspect_data_size method.

Network events

network module notifies a script subscribed by network.subscribe("script_name") about events happend within it.

If plugin didn't subscribe for network events, no events are sent by implementation, it's not an error.
After a connection has been established (network.connect()) a connection handler gets returned, this handler is used to comminicate with a remote side. By default, io async notifications are sent to a script subscribed by network.subscribe(), to change it use network.set_handler()
It's possible to inspect data without fetching them while receiving data, use network.inspect().
All methods throw exceptions in a case of errors, unless otherwise stated.
All internal resources are kept alive till a plugin gets removed (uninstalled).

Name of a network event: "network"

Structure of a network event:

```
{
    "event_type": "string",
    "event": {}
}
```

fields	type	description
event_type	string	type of a network event
event	table	data specific to a network event

Types of network events:

- I/O activity on connection handlers ("io_activity")

Structure of an io_activity event:

```
{
    "handle": 2810,
    "event": "IN"
}
```

fields	type	description
handle	int	handle an I/O activity happend on
event	string	type of an I/O activity

Type of an I/O activity:

- IN (there's some data on a handle to read without blocking)
- OUT (it's possible to write to a handle without blocking)
- CLOSED (remote side closed a connection)
- CONNECTED (connection has been established successfully)
- FAILED_TO_CONNECT (fail to establish a connection)

Network constans:

network.event_types:

```
{
    "io_activity": {
        "type_name": "io_activity",
        "IN": "IN",
        "OUT": "OUT",
        "CLOSED": "CLOSED",
        "CONNECTED": "CONNECTED",
        "FAILED_TO_CONNECT": "FAILED_TO_CONNECT"
    }
}
```

---

subscribe()

Subcribes a specified script for network events.
Subscribed script can be notified (called) concurrently with other scripts, it's up to a plugin's author to make a proper protection.

call: network.subscribe()

params: script name (string).

return: nothing

example:

```
require "network"

network.subscribe( "event_handling" )
```

---

connect()

Establishes a connection to a remote side with a specified address. It's an async call, when connection has been establised, an event CONNECTED or FAILED_TO_CONNECT gets sent to a set events handler.
Throw an exception if something local went wrong.
Both IPv4 and IPv6 along with a host name can be used as a remote address.
Once connection has been established there's no way to change its type (TCP, UDP) and an address it's connected to.

call: network.connect()

params: address (table), options (int) [optional].

Structure of an address parameter:

```
   {
      "ip": "string",
      "port": "string"
      "connection_type": "string"
    }
```

return: connection handle (int).

example:

```
require "network"

hndl = network.connect( { ip = "127.0.0.1", port = "6010", connection_type = "TCP" } )
if not hndl then
  print( "fail to connect" )
end
```

---

send()

Sends data to a remote side.
It's a plugin's author responsibility to make sure all data was sent (repeatedly calling this function till all data gets sent or, using I/O async notifiction, call this function when it exact has to be called).

call: network.send()

params: hndl (int), binary data (string).

return: amount of sent data (int)

example:

```
require "network"

hndl = network.connect( {} )
conn = network.send( hndl, "hello_world" )
```

---

receive()

Receives data from a remote side. It's an async call, never blocks, if there's no data to read, returns immediately.
It's a plugin's author responsibility to make sure all data was consumed (repeatedly calling this function till all data gets consumed or, using I/O async notifiction, call this function when it exact has to be called).
Fetch data from an internal buffer, see inspect_data to inspect data without fetching them.

call: network.receive()

params: hndl (int), amount of data to return, in bytes (int) [optional, by default receive will return all there is].

return: received data (string) [may be empty if there's no data to read]

example:

```
require "network"

hndl = network.connect( {} )
data = network.receive( hndl, 100 )
```

---

inspect_data()

Inspects received data on a connection handle without fetching them from an internal buffer. Makes a copy of an internal implementation buffer.

call: network.inspect_data()

params: hndl (int).

return: binary data (string)

example:

```
require "network"

hndl = network.connect( {} )
data = network.inspect_data( hndl )

if string.find( data, "OK" ) then
  print( "we got a specific chunk of data, do something useful..." )
end
```

---

inspect_data_size()

Inspects a size of received data, doesn't make a copy of an internal buffer;

call: network.inspect_data_size()

params: hndl (int).

return: size of received data, in bytes (int)

example:

```
require "network"

hndl = network.connect( {} )
amount = network.inspect_data_size( hndl )

if amount >= 10 then
  print( "we got enough data to start processing..." )
end
```

---

set_handler()

Set a custom events handler. If network.subscribe() wasn't called this's the only one way to get I/O async notifications for handler.
After this call network sends events for the handler only to a specified events handler.

call: network.set_handler()

params: hndl (int), events handler script (string).

return: nothing

example:

```
require "network"

hndl = network.connect( {} )
network.set_handler( hndl, "events_handler" )
```

---

close()

Close a connection handle. Implementation has to free all associated resources.

call: network.close()

params: hndl (int).

return: nothing

example:

```
require "network"

hndl = network.connect( {} )
network.close( hndl )
```

Timer module

Timer module description

The module provides functions (timers) which allow to delay the execution of arbitrary scripts. The **set_timeout** function is commonly used if desired to have function called once after the specified delay. The **set_interval** function is commonly used to set a delay for functions that are executed again and again.
The interval the timer will wait before executing its action may not be exactly the same as the interval specified by the user.

Timer module API (require "timer")

Functions

Return	Name and parameters
timer_id	**set_timeout**(*delay, script, args*)
none	**set_timeout_with_id**(*delay, timer_id, script, args*)
timer_id	**set_interval**(*delay, script, args*)
none	**set_interval_with_id**(*delay, timer_id, script, args*)
none	**cancel**(*timer_id*)
bool	**exists**(*timer_id*)

none	**reset**(*delay, timer_id*)

---

set_timeout

Calls a script after specified delay.

Syntax:

```
local timer_id = timer.set_timeout( delay, script, args )
```

Params:

fields	type	description
delay	number	The time, in milliseconds (thousandths of a second), the timer should wait before the specified script is executed. Note that the actual delay may be longer
script	string	A script to be executed after the timer expires
args	table	Parameters which are passed through to the script. Note **timeId** will be added to args table

Returns

fields	type	description
timer_id	string	ID which uniquely identifies the timer

Examples

```
local timer = require("timer")
```

```
local timer_id = timer.set_timeout(1000, "HUB:module_name", { arg_name = "arg_value"
})
```

---

set_timeout_with_id

Calls a script after specified delay.

Syntax:
timer.set_timeout_with_id(delay, timer_id, script, args)

Params:

fields	type	description
delay	number	The time, in milliseconds (thousandths of a second), the timer should wait before the specified script is executed. Note that the actual delay may be longer
timer_id	string	ID which uniquely identifies the timer (No timer will be created, if timer with specified timer_id exists)
script	string	A script to be executed after the timer expires
args	table	Parameters which are passed through to the script. Note **timeId** will be added to args table

Returns

fields	type	description
none		

Examples

```
local timer = require("timer")

timer.set_timeout_with_id(1000, "ID", "HUB:module_name", { arg_name = "arg_value" })
```

---

set_interval

Calls a script repeatedly, with a fixed time delay between each call.
Syntax
`local timer_id = timer.set_interval( delay, script, args )`

Params:

fields	type	description
delay	number	The time, in milliseconds (thousandths of a second), the timer should delay in between executions of the specified script. Note that the actual delay may be longer
script	string	A script to be executed every `delay` milliseconds
args	table	Parameters which are passed through to the script. Note **timeId** will be added to args table

Returns

fields	type	description
timer_id	string	ID which uniquely identifies the timer

Examples

```
local timer = require("timer")

local timer_id = timer.set_interval(1000, "HUB:module_name", { arg_name = "arg_value"
})
```

---

set_interval_with_id

Calls a script repeatedly, with a fixed time delay between each call.

Syntax:

timer.set_interval_with_id(delay, timer_id, script, args)

Params:

fields	type	description
delay	number	The time, in milliseconds (thousandths of a second), the timer should delay in between executions of the specified script. Note that the actual delay may be longer
timer_id	string	ID which uniquely identifies the timer (No timer will be created, if timer with specified timer_id exists)
script	string	A script to be executed every `delay` milliseconds
args	table	Parameters which are passed through to the script. Note **timeId** will be added to args table

Returns

fields	type	description
none		

Examples

```
local timer = require("timer")

timer.set_interval_with_id(1000, "ID", "HUB:module_name", { arg_name = "arg_value" })
```

---

cancel

Cancels a timer previously established by calling set_timeout or set_interval. Passing an invalid ID to cancel() silently does nothing; no exception is thrown.

Syntax:

timer.cancel(timer_id)

Params

fields	type	description
timer_id	string	The identifier of the timer desired to cancel

Returns

fields	type	description
none		

Examples

```
-- some_module.lua

local timer = require("timer")
```

```lua
local timer_id = timer.set_interval(1000, "HUB:handler", { arg_name = "arg_value" })

-- handler.lua

params = ...

local timer = require("timer")

print(params.arg_name)

timer.cancel(params.timerId)
```

---

exists

Check if timer established with specified ID

Syntax:

`timer.exists( timer_id )`

Params:

fields	type	description
timer_id	string	The identifier of the timer

Returns

fields	type	description
	bool	`true` if timer is established, `false` otherwise

Examples

```lua
-- module.lua
```

```lua
local timer = require("timer")

if timer.exists("timer_id") then
    print("timer exist")
else
    print("timer does not exist")
end
```

---

reset

Reset a timer previously established by calling **set_timeout** or **set_interval**. Passing an invalid ID to restart() silently does nothing; no exception is thrown.

Syntax:

timer.reset(delay, timer_id)

Params:

fields	type	description
delay	number	New delay for the timer. The time, in milliseconds (thousandths of a second). Note that the actual delay may be longer
timer_id	string	The identifier of the timer desired to restart

Returns:

fields	type	description
none		

Examples

```lua
-- module.lua
```

```
local timer = require("timer")

local timer_id = timer.set_interval(1000, "HUB:handler", { arg_name = "arg_value" })

timer.reset(2500, timer_id)
```

Zwave module

Zwave module description

Module provides an access to a zwave functionality (include/exclude devices, set values, get value, etc.).

Zwave module API (require "zwave")

reset()

Delete ALL data and restore Z-Wave chip to factory default.

call: zwave.reset()

params: none

return: none

example:

```
require "zwave"

zwave.reset()
```

start_include()

Start a procedure of including a new ZWave device. Cause a ZWave chip to send an "include" broadcast. If some ZWave device has been included a node_added event gets sent.

call: zwave.start_include()

params: none.

return: none.

example:

```
require "zwave"
```

```
    zwave.start_include()
```

---

start_exclude()

Start a procedure of excluding a previously included ZWave device. Cause a ZWave chip to send an "exclude" broadcast.

call: zwave.start_exclude()

params: none.

return: none.

example:

```
   require "zwave"

    zwave.start_exclude()
```

---

subscribe()

Subscribe a script for Zwave events. After subscribing the script will be launched for each event happens on a Zwave addon and information about this event will be passed as a parameter.

call: zwave.subscribe()

params: script name (string).

return: none.

example:

```
   require "zwave"

    zwave.subscribe( "HUB:zwave/scripts/events_handling" )
```

unsubscribe()

Unsubscribe a script from Zwave events.

call: zwave.unsubscribe()

params: script name (string).

return: none.

example:

```
require "zwave"

zwave.unsubscribe( "HUB:zwave/scripts/events_handling" )
```

set_device_specific_key()

Enter S2 device specific key (response)

call: zwave.set_device_specific_key()

params: device specific key (private part, 1 number)

return: none.

example:

```
require "zwave"

zwave.set_device_specific_key( 34338 )
```

set_auth_mode()

Select authentication modes (response)

call: zwave.set_auth_mode()

params: selected modes (table)

return: none.

example:

```
require "zwave"

local modes = { "accessControl", "unauthenticated" }
zwave.set_auth_mode( modes )
```

stop_include()

Stops a procedure of including a new ZWave device. Cause a ZWave chip to send an "include" broadcast.

call: zwave.stop_include()

params: none.

return: none.

example:

```
require "zwave"

zwave.stop_include()
```

stop_security_include()

Stops a procedure of including S2 security device in decured mode. Device will be included in non-secure mode.

call: zwave.stop_security_include()

params: none.

return: none.

example:

```
require "zwave"

zwave.stop_security_include()
```

get_all_nodes_ids()

Get list of ZWave node ids included into network.

call: zwave.get_all_nodes_ids()

params: none.

return: list of ZWave nodes.

example:

```
require "zwave"

local node_ids = zwave.get_all_nodes_ids()
for _, node_id in ipairs(node_ids) do
    local node = zwave.get_node( node_id )
    if node then do
        print( "node.is_reachable: " .. node.isReachable )
    end
end
```

get_node()

Get a ZWave node by a node_id. (node_id is a part of data coming to a script subscribed for ZWave events)

call: zwave.get_node()

params: node id (number).

return: ZWave node (table). Look at objects page to get a node table format.

example:

```
require "zwave"

local node = zwave.get_node( node_id )
if node then do
    print( "node.is_reachable: " .. node.isReachable )
end
```

get_command_class()

Get a ZWave command class by node id, channel id and command class id. (node_id, channel_id, cc_id is a part of data coming to a script subscribed for ZWave events)

call: zwave.get_command_class()

params: node_id (int), channel_id (int), class_id (int)

return: ZWave command class (table). Look at objects page to get a command classes table format.

example:

```
require "zwave"

local cc = zwave.get_command_class( node_id, channel_id, cc_id )
if cc then do
    print( "cc.version: " .. cc.version )
end
```

set_value()

Set a specific value for a specified ZWave pcp (pcp is specified by node_id, class_id and channel_id).

call: zwave.set_value()

params: node_id (int), class_id (int), channel_id (int), value (boolean or number). return: none.

example:

```
require "zwave"

value = true
zwave.set_value( node_id, class_id, channel_id, value )
```

set_value_multicast()

Set a specific value for a specified list of ZWave pcp's (pcp's are specified by node_ids, class_id and channel_id).

call: zwave.set_value_multicast()

params: node_ids (array of int's), class_id (int), channel_id (int), value (boolean or number). return: none.

example:

```
require "zwave"

value = true
zwave.set_value_multicast( {node_id_1, node_id_2}, class_id, channel_id, value )
```

set_token_value()

Set a specific token value for a specified ZWave pcp (pcp is specified by node_id, class_id and channel_id).

call: zwave.set_token_value()

params: node_id (int), class_id (int), channel_id (int), value (string). return: none.

example:

```
require "zwave"

value = "token"
zwave.set_token_value( node_id, class_id, channel_id, value )
```

---

set_token_value_multicast()

Set a specific token value for a specified list of ZWave pcp's (pcp's are specified by node_ids, class_id and channel_id).

call: zwave.set_token_value_multicast()

params: node_ids (array of int's), class_id (int), channel_id (int), value (string). return: none.

example:

```
   require "zwave"

    value = "token"
    zwave.set_token_value_multicast( {node_id_1, node_id_2}, class_id, channel_id,
value )
```

set_rgb_value()

Set a specific rgb value for a specified ZWave pcp (pcp is specified by node_id, class_id and channel_id).

call: zwave.set_rgb_value()

params: node_id (int), class_id (int), channel_id (int), value (rgb). return: none.

example:

```
   require "zwave"

    value = { red = 0x08, green = 0x11, blue = 0x33 }
    zwave.set_rgb_value( node_id, class_id, channel_id, value )
```

label	value
wwhite	0x00 – 0xFF: 0 – 100%
cwhite	0x00 – 0xFF: 0 – 100%
red	0x00 – 0xFF: 0 – 100%
green	0x00 – 0xFF: 0 – 100%

blue	0x00 – 0xFF: 0 – 100%
amber	0x00 – 0xFF: 0 – 100%
cyan	0x00 – 0xFF: 0 – 100%
purple	0x00 – 0xFF: 0 – 100%

set_rgb_value_multicast()

Set a specific rgb value for a specified list of ZWave pcp's (pcp's are specified by node_ids, class_id and channel_id).

call: zwave.set_rgb_value_multicast()

params: node_ids (array of int's), class_id (int), channel_id (int), value (rgb).

return: none.

example:

```
   require "zwave"

    value = { red = 0x08, green = 0x11, blue = 0x33 }
    zwave.set_rgb_value_multicast( {node_id_1, node_id_2}, class_id, channel_id, value
)
```

label	value
wwhite	0x00 – 0xFF: 0 – 100%
cwhite	0x00 – 0xFF: 0 – 100%

red	0x00 – 0xFF: 0 – 100%
green	0x00 – 0xFF: 0 – 100%
blue	0x00 – 0xFF: 0 – 100%
amber	0x00 – 0xFF: 0 – 100%
cyan	0x00 – 0xFF: 0 – 100%
purple	0x00 – 0xFF: 0 – 100%

set_sound_switch_configuration()

Set default volume and tone ID for a ZWave node

call: zwave.set_sound_switch_configuration()

params: node_id (int), class_id (int), channel_id (int), volume (int), default_tone_id (int).

return: none.

example:

```
require "zwave"

zwave.set_sound_switch_configuration( node_id, class_id, channel_id, 50, 2 )
```

request_sound_switch_configuration()

Send a request for default volume and tone ID to a ZWave node

call: zwave.request_sound_switch_configuration()

params: none. return: none.

example:

```
require "zwave"

zwave.request_sound_switch_configuration( node_id, class_id, channel_id )
```

set_sound_switch_tone_play()

Set current tone ID and volume for a ZWave node

call: zwave.set_sound_switch_tone_play()

params: node_id (int), class_id (int), channel_id (int), tone_id (int), volume (int).

return: none.

example:

```
require "zwave"

zwave.set_sound_switch_tone_play( node_id, class_id, channel_id, 3, 50 )
```

request_sound_switch_tone_play()

Send a request for current tone ID and volume to a ZWave node

call: zwave.request_sound_switch_tone_play()

params: none. return: none.

example:

```
require "zwave"

zwave.request_sound_switch_tone_play( node_id, class_id, channel_id )
```

set_door_lock_mode()

Set a specific door lock mode for a ZWave node

call: zwave.set_door_lock_mode()

params: node_id (int), channel_id (int), value (string).

return: none.

example:

```
   require "zwave"

    zwave.set_door_lock_mode( node_id, channel_id, "secured" )
```

set_setpoint_value()

Set a setpoint value for a specified ZWave pcp (pcp is specified by node_id, class_id, channel_id, and setpoint_type).

call: zwave.set_setpoint_value()

params: node_id (int), class_id (int), channel_id (int), setpoint_type (int), scale (int), value (int).

return: none.

example:

```
   require "zwave"

    node_id = 1
    class_id = 67
    channel_id = 0
    setpoint_type = 1
    scale = 0
    value = 24

    zwave.set_setpoint_value( node_id, class_id, channel_id, setpoint_type, scale,
value )
```

set_configuration_value()

Set a configuration value for a specified ZWave pcp (pcp is specified by node_id, channel_id, and configuration number).

call: zwave.set_configuration_value()

params: node_id (int), channel_id (int), param_number (int), param_size (int), value (int).

return: none.

example:

```
   require "zwave"

    node_id = 1
    channel_id = 0
    param_number = 1
    param_size = 2
    value = 24

    zwave.set_configuration_value( node_id, channel_id, param_number, param_size,
value )
```

reset_configuration_value()

Reset a configuration value for a specified ZWave pcp to its default (pcp is specified by node_id, channel_id, and configuration number).

call: zwave.reset_configuration_value()

params: node_id (int), channel_id (int) and param_number (int). return: none.

example:

```
   require "zwave"

    node_id = 1
    channel_id = 0
    param_number = 1

    zwave.reset_configuration_value( node_id, channel_id, param_number )
```

switch_multilevel_level_change()

This command used to initiate a transition to a new level or stop an ongoing transition.

call: zwave.switch_multilevel_level_change()

params: node_id (int), channel_id (int), command (string). return: none.

example:

```
require "zwave"

node_id = 2
channel_id = 0
command = "up"

zwave.switch_multilevel_level_change( node_id, channel_id, command )
```

request_value()

Ask the ZWave addon to send a *_updated event for a specified pcp, that is get a pcp value asynchronously, by event. (pcp is specified by node_id, class_id and channel_id).

call: zwave.request_value()

params: node_id (int), class_id (int), channel_id (int).

return: none.

example:

```
require "zwave"

zwave.request_value( node_id, class_id, channel_id, value )
```

request_node_values()

Ask the ZWave addon to send a *_updated event for all pcp's, that is get a pcp's values asynchronously, by event.
Before each request the delay should be expired. Note that request_delay parameter currently used only for door lock and schedule entry lock command classes, and actual delay may be longer

call: zwave.request_node_values()

params: node_id (int) - id of node in a current znet mesh (1 - 232). request_delay (int), the time in milliseconds (thousandths of a second).

return: none.

example:

```
require "zwave"

zwave.request_node_values( node_id, 1000 )
```

request_setpoint_value()

Ask the ZWave addon to send a thermostat_setpoint_value_updated event for a specified pcp, that is get a pcp value asynchronously, by event. (pcp is specified by node_id, channel_id, and setpoint_type).

call: zwave.request_setpoint_value()

params: node_id (int), channel_id (int), setpoint_type (int).

return: none.

example:

```
require "zwave"

zwave.request_setpoint_value( node_id, channel_id, setpoint_type )
```

request_user_code()

Ask ZWave addon to send a 'USER_CODE_GET' request to ZWave node, user_code_updated event will be sent as a request result

call: zwave.request_user_code()

params: node_id (int), channel_id (int), user_id (int)(optional). If user_id is not specified, all supported user codes will be requested.

return: none.

example:

```
require "zwave"

zwave.request_user_code(node_id, channel_id, user_id)
```

request_configuration_value()

Ask the ZWave addon to send a configuration_value_updated event for a specified pcp, that is get a pcp configuration value asynchronously, by event. (pcp is specified by node_id, channel_id and param_number).

call: zwave.request_configuration_value()

params: node_id (int), channel_id (int), param_number (int).

return: none.

example:

```
require "zwave"

zwave.request_configuration_value( node_id, channel_id, param_number )
```

request_association_group()

Ask ZWave addon to send a 'ASSOCIATION_GET' or 'MULTI_CHANNEL_ASSOCIATION_GET' request to ZWave node, association_updated event will be sent as a request result

call: zwave.request_user_code()

params: node_id (int), channel_id (int), group_id (int).

return: none.

example:

```
require "zwave"
```

```
    zwave.request_association_group(node_id, channel_id, group_id)
```

add_node_to_association_group()

Ask ZWave addon to add node to association group for ZWave node

call: zwave.add_node_to_association_group()

params: node_id (int), channel_id (int), group_id (int), destinationNode(int), destinationEndpoint (int)(optional)

return: none.

example:

```
   require "zwave"

    zwave.add_node_to_association_group(node_id, channel_id, group_id,
destinationNode, destinationEndpoint)
```

remove_node_from_association_group()

Ask ZWave addon to remove node from association group for ZWave node

call: zwave.remove_node_from_association_group()

params: node_id (int), channel_id (int), group_id (int), destinationNode(int), destinationEndpoint (int)(optional)

return: none.

example:

```
   require "zwave"

    zwave.remove_node_from_association_group(node_id, channel_id, group_id,
destinationNode, destinationEndpoint)
```

set_user_code()

Ask ZWave addon to set user code for ZWave node

call: zwave.set_user_code()

params: node_id (int), channel_id (int), user_id (int), user_id_status (string), user_code (string)

return: none.

example:

```
require "zwave"

zwave.set_user_code(node_id, channel_id, user_id, user_id_status, user_code)
```

set_silent_sensors()

Sets set of sensors to silent mode. The rest sensors of current node and channel become audible.

call: zwave.set_silent_sensors()

params:

- node_id (int),
- channel_id (int),
- sound_delay (int) - 16 bit, seconds.
- sensor_names (string array) - sensor (or notification) types to be silent.

return: none.

example:

```
require "zwave"

zwave.set_silent_sensors(node_id, channel_id, 0, { "co_alarm", "smoke_alarm" })
```

request_alarm_sensor_value()

Requests current state

call: zwave.request_alarm_sensor_value()

params:

- node_id (int),
- channel_id (int),
- sensor_name (string) - see alarm sensor types

return: none.

example:

```
require "zwave"

 zwave.request_alarm_sensor_value(node_id, channel_id, "smoke_alarm")
```

set_week_day_entry_lock_schedule

Sets week day entry lock schedule for specific node, channel, user, slot

call: zwave.set_week_day_entry_lock_schedule()

params:

- node_id (int),
- channel_id (int),
- user_id (int), - select one of existing user (see user code cc)
- slot_id (int),
- week_day_entry_lock_schedule (table) - optional. Pass nil for removimg

return: none.

example:

```
require "zwave"

 zwave.set_week_day_entry_lock_schedule(node_id, channel_id, user_id, slot_id,
     {
         weekDay = "monday",
         startHour = 11,
         startMinute = 20,
         stopHour = 14,
         stopMinute = 10
```

```
        })
```

request_week_day_entry_lock_schedule

Requests week day entry lock schedule for specific node, channel, user, slot

call: zwave.request_week_day_entry_lock_schedule()

params:

- node_id (int),
- channel_id (int),
- user_id (int), - select one of existing user (see user code cc)
- slot_id(int)

return: none

example:

```
require "zwave"

zwave.request_week_day_entry_lock_schedule(node_id, channel_id, user_id, slot_id)
```

set_daily_repeating_entry_lock_schedule

Sets daily repeating entry lock schedule for specific node, channel, user, slot

call: zwave.set_daily_repeating_entry_lock_schedule()

params:

- node_id (int),
- channel_id (int),
- user_id (int), - select one of existing user (see user code cc)
- slot_id(int),
- daily_repeating_entry_lock_schedule(table), - optional. Pass nil for removimg

return: none.

example:

```
require "zwave"
```

```
    zwave.set_daily_repeating_entry_lock_schedule(node_id, channel_id, user_id,
slot_id,
        {
            weekDays = { "monday", "friday" },
            startHour = 11,
            startMinute = 20,
            durationHour = 14,
            durationMinute = 50
        })
```

request_daily_repeating_entry_lock_schedule

Requests daily repeating entry lock schedule for specific node, channel, user, slot

call: zwave.request_daily_repeating_schedule()

params:

- node_id (int),
- channel_id (int),
- user_id (int), - select one of existing user (see user code cc)
- slot_id(int)

return: none.

example:

```
   require "zwave"

    zwave.request_daily_repeating_schedule(node_id, channel_id, user_id, slot_id)
```

set_year_day_entry_lock_schedule

Sets year day entry lock schedule for specific node, channel, user, slot

call: zwave.set_year_day_entry_lock_schedule()

params:

- node_id (int),
- channel_id (int),
- user_id (int), - select one of existing user (see user code cc)

- slot_id(int),
- year_day_entry_lock_schedule(table), - optional. Pass nil for removimg

return: none.

example:

```
   require "zwave"

    zwave.set_year_day_entry_lock_schedule(node_id, channel_id, user_id, slot_id,
        {
            startYear = 2019,
            startMonth = 1,
            startDay = 10,
            startHour = 23,
            startMinute = 1,
            stopYear = 2019,
            stopMonth = 2,
            stopDay = 10,
            stopHour = 23,
            stopMinute = 1,
        })
```

request_year_day_entry_lock_schedule

Requests year day entry lock schedule for specific node, channel, user, slot

call: zwave.request_year_day_entry_lock_schedule()
params:

- node_id (int),
- channel_id (int),
- user_id (int), - select one of existing user (see user code cc)
- slot_id(int)

return: none.

example:

```
   require "zwave"

    zwave.request_year_day_entry_lock_schedule(node_id, channel_id, user_id, slot_id)
```

ZWave events

The ZWave addon notifies a script subscribed by `zwave.subscribe( "script_name" )` about state of the ZWave world. Structure of an event is table.

Type of events:

---

module_reset - gets sent if a ZWave chip has been restored to factory default

```
{
    event = "module_reset"
    status = "started"
}
```

fields	type	description
event	string	type of an event
status	string	actual **event status**

---

include_invoked - gets sent to notify of device inclusion operation tried to start. This event is initiated before the start_include method

```
{
    event = "include_invoked"
}
```

fields	type	description
event	string	type of an event

---

include_started - gets sent to notify of device inclusion operation start. This event is initiated by the start_include method

```
{
    event = "include_started"
}
```

fields	type	description
event	string	type of an event

---

include_finished - gets sent to notify of device inclusion operation completion. This event is initiated by the start_include method

```
{
    event = "include_finished"
}
```

fields	type	description
event	string	type of an event

---

include_finished_error - gets sent to notify of device inclusion operation failed. This event is initiated by the start_include method

```
{
    event = "include_finished_error"
}
```

fields	type	description

event	string	type of an event

---

include_finished_timeout - gets sent to notify of device inclusion operation took a lot of time. This event is initiated by the start_include method

```
{
    event = "include_finished_timeout"
}
```

fields	type	description
event	string	type of an event

---

exclude_invoked - gets sent to notify of device exclusion operation tried to start. This event is initiated before the start_exclude method

```
{
    event = "exclude_invoked"
}
```

fields	type	description
event	string	type of an event

---

exclude_started - gets sent to notify of device exclusion operation start. This event is initiated by the start_exclude method

```
{
    event = "exclude_started"
}
```

fields	type	description
event	string	type of an event

---

exclude_finished - gets sent to notify of device exclusion operation completion. This event is initiated by the start_exclude method

```
{
    event = "exclude_finished"
}
```

fields	type	description
event	string	type of an event

---

exclude_finished_error - gets sent to notify of device exclusion operation failed. This event is initiated by the start_exclude method

```
{
    event = "exclude_finished_error"
}
```

fields	type	description
event	string	type of an event

---

exclude_finished_timeout - gets sent to notify of device exclusion operation took a lot of time. This event is initiated by the start_exclude method

```
{
    event = "exclude_finished_timeout"
}
```

fields	type	description
event	string	type of an event

---

action_busy - gets sent to notify that addon can not run new request, because it is busy by other action.

```
{
    event = "action_busy",
    action_type = "include_starting"
}
```

fields	type	description
event	string	type of an event
action_type	string	actual **action**, which is running

---

binary_sensor_value_updated - gets sent to notify that binary sensor value has been changed

```
{
    event = "binary_sensor_value_updated",
    nodeId = 1,
    classId = 48,
    channelId = 0,
    value = 1,
    sensorType = 5
}
```

fields	type	description

event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	ZWave command class id (always 48 for this type of event)
channelId	int	channel id
value	uint	0 or 1 (meaning depends on sensor type)
sensorType	uint	binary sensor type according to ZWave specification

---

configuration_value_updated - result of configuration value request (see **request_configuration_value** api)

```
{
    event = "configuration_value_updated",
    nodeId = 1,
    classId = 48,
    channelId = 0,
    parameter_num = 1,
    parameter_size = 2,
    value = 1
}
```

fields	type	description
event	string	type of an event

nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	ZWave command class id (always 112 for this type of event)
channelId	int	channel id
parameter_num	uint	configuration parameter number
parameter_size	uint	configuration parameter value size
value	uint	value of configuration parameter

---

door_lock_state_updated - gets sent to notify that door lock state was updated

```
{
    event = "door_lock_state_updated",
    nodeId = 1,
    channelId = 0,
    doorMode = "unsecured",
    doorCondition = {
        latchState = "latch_open",
        boltState = "bolt_unlocked",
        doorState = "door_open"
    }
}
```

fields	type	description
event	string	type of an event

nodeId	int	a unique id of a ZWave device within a Zwave network
channelId	int	channel id
doorMode	string	actual **door lock mode**
doorCondition.latchStat e	string	actual **latch state**
doorCondition.boltState	string	actual **bolt state**
doorCondition.doorState	string	actual **door state**

---

multilevel_sensor_value_updated - gets sent to notify that multilevel sensor value has been changed

```
{
    event = "multilevel_sensor_value_updated",
    nodeId = 1,
    classId = 49,
    channelId = 0,
    value = 10,
    scale = "lux",
    sensorType = "illuminance"
}
```

fields	type	description
event	string	type of an event

nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	ZWave command class id (always 49 for this type of event)
channelId	int	channel id
value	float	value
scale	string	value scale (one of supported for current item value type)
sensorType	string	multilevel sensor type according to ZWave specification

---

node_added - gets sent if a new ZWave device has been included

```
{
    event = "node_added",
    nodeId = 2
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network

---

reachable_state_updated - gets sent if a ZWave device reachable state has been changed

```
{
    event = "reachable_state_updated",
```

```
    nodeId = 2,
    value = true
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
value	bool	`true` if ZWave device is reachable, `false` otherwise

---

s2_select_authentication_modes - gets sent to informs about starting of S2 flow and propose available authentication modes

```
{
    event = "s2_select_authentication_modes",
    modes = { "accessControl", "authenticated", "unauthenticated" }
}
```

fields	type	description
event	string	type of an event
modes	table	an array of available **authentication modes**

List of possible authentication modes:

Mode	description

accessControl	S2 Access Control
authenticated	S2 Authenticated
unauthenticated	S2 Unauthenticated
s0	S0
clientSide	Client-side authentication

---

s2_show_device_side_key - gets sent to show device side key

```
{
    event = "s2_show_device_side_key",
    key = { 58975, 37056 }
}
```

fields	type	description
event	string	type of an event
key	table	an array of 2 items device specific key

---

s2_request_device_specific_key - gets sent to request device specific key

```
{
    event = "s2_request_device_specific_key",
    key = { 58975, 37056, 29754, 11588, 18902, 12511, 34338 }
}
```

fields	type	description
event	string	type of an event
key	table	an array of 7 items device specific key (public part)

---

node_removed - gets sent if a ZWave device has been excluded from hub network

```
{
    event = "node_removed",
    nodeId = 2,
    manufacturedId = 134,
    productId = 2,
    productType = 10
}
```

fields	type	description
event	string	type of an event
nodeId	int	
manufacturedId	int	
productId	int	
productType	int	

---

unknown_node_removed - gets sent if a unknown ZWave device has been excluded from another network

```
{
    event = "unknown_node_removed"
}
```

fields	type	description
event	string	type of an event

---

value_updated - gets sent if a pcp value has been changed

```
{
    event = "value_updated",
    nodeId = 2,
    classId = 37,
    channelId = 0,
    value = 0
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	

value	int

---

token_value_updated - gets sent if a pcp token value has been changed

```
{
    event = "token_value_updated",
    nodeId = 2,
    classId = 37,
    channelId = 0,
    value = "token"
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	channel id
value	string	token value

---

user_code_updated - result of user code request (see **request_user_code** api)

```
{
    event = "user_code_updated",
    nodeId = 2,
    channelId = 0,
    userId = 2,
    supportedUsersCount = 250,
```

```
    userIdStatus = "available",
    userPollingStatus = "polling",
    userCode = "5482"
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
channelId	int	channel id
userId	int	user id (from 1 to number of supported users)
supportedUsersCount	int	user id's count (from 1 to 250)
userIdStatus	string	actual **user id status**
userPollingStatus	string	actual **user polling status**
userCode	string	user code (empty or min 4 digits code)

---

thermostat_setpoint_value_updated - gets sent if a thermostat setpoint value has been changed

```
{
    event = "thermostat_setpoint_value_updated",
    nodeId = 2,
    classId = 67,
```

```
    channelId = 0,
    type = 1,
    scale = 0,
    value = 11
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	
type	int	type of an setpoint
scale	int	temperature scale
value	int	actual setpoint value

---

thermostat_operating_state_value_updated - gets sent if a thermostat operating state value has been changed

```
{
    event = "thermostat_operating_state_value_updated",
    nodeId = 2,
    classId = 66,
    channelId = 0,
    value = "heating"
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	
value	string	actual **operating state**

---

float_value_updated - gets sent if a float pcp value has been changed

```
{
    event = "float_value_updated",
    nodeId = 2,
    classId = 37,
    channelId = 0,
    value = 0,
    type = 1,
    scale = 1,
    precision = 3
}
```

fields	type	description
event	string	type of an event

nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	
value	int	
type	int	type of metering physical unit
scale	int	a value of measurement unit
precision	int	a number of decimal places of the value

---

button_state_updated - gets sent if a pcp button state has been changed

```
{
    event = "button_state_updated",
    nodeId = 2,
    classId = 37,
    channelId = 0,
    buttonId = 1,
    value = "press_1_time"
}
```

fields	type	description
event	string	type of an event

nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	channel id
buttonId	int	button id
value	string	button state

---

association_updated - gets sent if a pcp association group nodes have been changed

```
{
    event = "association_updated",
    nodeId = 2,
    classId = 58,
    channelId = 0,
    groupId = 1,
    groupNodes = [ {nodeId = 1, endpointId = 0} ]
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device

channelId	int	channel id
groupId	int	id of association group
groupNodes	table	array of endpoints in group(nodeId and endpointId for each endpoint)

---

scene_activated - gets sent if a scene activeted using scene controller

```
{
    event = "scene_activated",
    nodeId = 2,
    classId = 43,
    channelId = 0,
    sceneId = 1
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	channel id
sceneId	int	scene id

---

example: (event_handling.lua)

```
   local params = ...

    if params.event == "node_added" then
        print( "we got a node_added event for a " .. params.nodeId .. " device" )
    elseif params.event == "value_updated" then
        print( "we got a value_updated event, pcp: " .. to_string( params.classId ) +
to_string( params.channelId ) )
    end
```

---

hail - gets sent if a hail has been received

```
{
    event = "hail",
    nodeId = 2,
    classId = 130
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device

---

alarm_sensor_value_updated - gets sent if alarm sensor value has been changed

```
{
    event = "alarm_sensor_value_updated",
    nodeId = 2,
    classId = 156,
    channelId = 0,
    sensorType = "water_alarm",
    value = 255,
    delay = 0
```

```
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	classID + channelId construct a unique id of a pcp for a ZWave device
channelId	int	
sensorType	string	One of alarm sensor types
value	int	alarm sensor value. 0 - no alarm; 255 - alarm; 1-100 - severity in percents.
delay	int	indicates time in seconds the remote alarm must be active since last received report

---

week_day_entry_lock_schedule_updated - gets sent if week day entry lock schedule updated (or removed)

```
{
    event = "week_day_entry_lock_schedule_updated",
    nodeId = 2,
    classId = 78,
    channelId = 0,
    value =
```

```
    {
        weekDay = "monday",
        startHour = 11,
        startMinute = 20,
        stopHour = 14,
        stopMinute = 10
    }
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	
channelId	int	
userId	int	id of user (see user code cc for details)
slotId	int	
value	table	object of type, optional

---

daily_repeating_entry_lock_schedule_updated - - gets sent if daily repeating entry lock schedule updated (or removed)

```
{
    event = "daily_repeating_entry_lock_schedule_updated",
    nodeId = 2,
    classId = 78,
```

```
    channelId = 0,
    userId = 3,
    slotId = 1,
    value =
    {
        weekDays = { "monday", "friday" },
        startHour = 11,
        startMinute = 20,
        durationHour = 14,
        durationMinute = 50
    }
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	
channelId	int	
userId	int	id of user (see user code cc for details)
slotId	int	
value	table	object of type, optional

---

year_day_entry_lock_schedule_updated - - gets sent if year day entry lock schedule updated (or removed)

```
{
    event = "year_day_entry_lock_schedule_updated",
    nodeId = 2,
    classId = 78,
    channelId = 0,
    userId = 3,
    slotId = 1,
    value =
    {
        startYear = 2019,
        startMonth = 1,
        startDay = 10,
        startHour = 23,
        startMinute = 1,
        stopYear = 2019,
        stopMonth = 2,
        stopDay = 10,
        stopHour = 23,
        stopMinute = 1
    }
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	
channelId	int	
userId	int	id of user (see user code cc for details)

slotId	int	
value	table	object of type, optional

---

sound_switch_configuration_updated - - gets sent if sound switch configuration is changed

```
{
    event = "sound_switch_configuration_updated",
    nodeId = 2,
    classId = 78,
    channelId = 0,
    defaultToneId = 1,
    volume = 100
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Zwave network
classId	int	
channelId	int	
defaultToneId	int	
volume	int	

---

sound_switch_tone_play_updated - - gets sent if sound switch tone being played is changed

```
{
    event = "sound_switch_tone_play_updated",
    nodeId = 2,
    classId = 78,
    channelId = 0,
    toneId = 1,
    volume = 100
}
```

fields	type	description
event	string	type of an event
nodeId	int	a unique id of a ZWave device within a Z-Wave network
classId	int	
channelId	int	
toneId	int	
volume	int	

ZWave Values

User id statuses

Values
`available`

`occupied`
`administrator_reserved`
`not_available`

User polling statuses

Values
`idle`
`polling`

S2 authentication modes

Values
`accessControl`
`authenticated`
`unauthenticated`
`s0`

`clientSide`

Thermostat operating states

Values
`idle`
`heating`
`cooling`
`fan_only`
`pending_heat`
`pending_cool`
`vent_economizer`
`aux_heating`
`2nd_stage_heating`
`2nd_stage_cooling`

`2nd_stage_aux_heat`
`3rd_stage_aux_heat`

Door lock modes

Values
`unsecured`
`unsecured_with_timeout`
`unsecured_for_inside`
`unsecured_for_inside_with_timeout`
`unsecured_for_outside`
`unsecured_for_outside_with_timeout`
`unknown`
`secured`

Door condition

Latch states

Values
`latch_open`
`latch_closed`

Bolt states

Values
`bolt_locked`
`bolt_unlocked`

Door states

Values
`door_open`
`door_closed`

Event statuses

Values
`invoked`
`started`
`failed`
`timeout`
`finished`

Action types

Values
`initializing`
`include_starting`
`include_stopping`
`exclude_starting`

`exclude_stopping`
`learn_starting`
`learn_stopping`
`monitoring`
`nif_sending`
`node_adding`
`node_removing`
`node_learning`
`node_list_getting`
`values_getting`
`reseting`
`idle`

Command

The value is used to set command in switch_multilevel_level_change command

Values
`up`
`down`
`stop`

Alarm sensor types

Note, alarm sensor cc is deprecated, notification cc is recommended extension of alarm sensor. Alarm sensor types (apart "general") are the same as first 5 notification types. Alarm sensor cc contains only alarm sensor types. If notification cc + alarm silence cc are on device, notification types can be used as extended alarm sensor types.

Values
`general`
`smoke_alarm`
`co_alarm`
`co2_alarm`
`heat_alarm`

`water_alarm`

Notification types

Values
`smoke_alarm`
`co_alarm`
`co2_alarm`
`heat_alarm`
`water_alarm`
`access_control`
`home_security`
`power_management`
`system`
`emergency_alarm`

clock
appliance
home_health
siren
water_valve
weather_alarm
irrigation
gas_alarm
pest_control
light_sensor
water_quality_monitoring
home_monitoring

Color Switch Component IDs

Values
`wwhite`
`cwhite`
`red`
`green`
`blue`
`amber`
`cyan`
`purple`
`indexed`

Zwave lua objects

Node

Node description.

```
{
```

```
    nodeId = "1",
    manufacturerId = 1,
    productType = 1,
    productId = 1,
    genericType = "GENERIC_TYPE_SWITCH_MULTILEVEL",
    basicType = 1,
    specificType = "SPECIFIC_TYPE_NOT_USED",
    security = { "accessControl", "authenticated", "unauthenticated", "s0" },
    securityClassStarted = { "accessControl", "authenticated", "unauthenticated", "s0"
},
    isReachable = 1,
    isBatteryPowered = 1,
    iconType = "ICON_TYPE_GENERIC_LIGHT_DIMMER_SWITCH",
    channels = {
        {
            id = 1,
            classes = {
                {
                    id = 1,
                    version = 1,
                    ...
                },
                {
                    id = 2,
                    version = 1,
                    ...
                },
            }
        },
        {
            id = 2,
            classes = {
                ...
            }
        },
        ...
    }
}
```

field	type	description
nodeId	int	

manufacturerId	int
productType	int
productId	int
genericType	int
basicType	int
security	table
securityClassStarted	table
isReachable	bool
isBatteryPowered	bool
channels	table
channels.id	int
classes	table
classes.id	int

version	int

Supported generic types:

Generic Type
GENERIC_TYPE_SWITCH_MULTILEVEL
GENERIC_TYPE_SWITCH_BINARY
GENERIC_TYPE_SWITCH_REMOTE
GENERIC_TYPE_SWITCH_TOGGLE
GENERIC_TYPE_THERMOSTAT
GENERIC_TYPE_ENTRY_CONTROL
GENERIC_TYPE_WINDOW_COVERING
GENERIC_TYPE_GENERIC_CONTROLLER
GENERIC_TYPE_WALL_CONTROLLER
GENERIC_TYPE_STATIC_CONTROLLER

GENERIC_TYPE_REPEATER_SLAVE
GENERIC_TYPE_NETWORK_EXTENDER
GENERIC_TYPE_AV_CONTROL_POINT
GENERIC_TYPE_APPLIANCE
GENERIC_TYPE_SENSOR_NOTIFICATION
GENERIC_TYPE_SECURITY_PANEL

Supported specific types:

Specific Type
SPECIFIC_TYPE_POWER_SWITCH_MULTILEVEL
SPECIFIC_TYPE_POWER_SWITCH_BINARY
SPECIFIC_TYPE_NOT_USED
SPECIFIC_TYPE_SCENE_SWITCH_MULTILEVEL
SPECIFIC_TYPE_FAN_SWITCH

SPECIFIC_TYPE_SWITCH_REMOTE_MULTILEVEL
SPECIFIC_TYPE_SWITCH_REMOTE_TOGGLE_MULTILEVEL
SPECIFIC_TYPE_SWITCH_TOGGLE_MULTILEVEL
SPECIFIC_TYPE_COLOR_TUNABLE_MULTILEVEL
SPECIFIC_TYPE_COLOR_TUNABLE_BINARY
SPECIFIC_TYPE_SCENE_SWITCH_BINARY
SPECIFIC_TYPE_SWITCH_REMOTE_BINARY
SPECIFIC_TYPE_SWITCH_REMOTE_TOGGLE_BINARY
SPECIFIC_TYPE_SWITCH_TOGGLE_BINARY
SPECIFIC_TYPE_VALVE_OPEN_CLOSE
SPECIFIC_TYPE_IRRIGATION_CONTROLLER
SPECIFIC_TYPE_THERMOSTAT_GENERAL
SPECIFIC_TYPE_THERMOSTAT_GENERAL_V2

SPECIFIC_TYPE_THERMOSTAT_HEATING
SPECIFIC_TYPE_DOOR_LOCK
SPECIFIC_TYPE_ADVANCED_DOOR_LOCK
SPECIFIC_TYPE_SECURE_KEYPAD_DOOR_LOCK
SPECIFIC_TYPE_SECURE_KEYPAD_DOOR_LOCK_DEADBOLT
SPECIFIC_TYPE_SECURE_LOCKBOX
SPECIFIC_TYPE_CLASS_A_MOTOR_CONTROL
SPECIFIC_TYPE_CLASS_B_MOTOR_CONTROL
SPECIFIC_TYPE_CLASS_C_MOTOR_CONTROL
SPECIFIC_TYPE_MOTOR_MULTIPOSITION
SPECIFIC_TYPE_SIMPLE_WINDOW_COVERING
SPECIFIC_TYPE_PORTABLE_REMOTE_CONTROLLER
SPECIFIC_TYPE_PORTABLE_SCENE_CONTROLLER

SPECIFIC_TYPE_PORTABLE_INSTALLER_TOOL
SPECIFIC_TYPE_REMOTE_CONTROL_AV
SPECIFIC_TYPE_REMOTE_CONTROL_SIMPLE
SPECIFIC_TYPE_BASIC_WALL_CONTROLLER
SPECIFIC_TYPE_PC_CONTROLLER
SPECIFIC_TYPE_SCENE_CONTROLLER
SPECIFIC_TYPE_STATIC_INSTALLER_TOOL
SPECIFIC_TYPE_SET_TOP_BOX
SPECIFIC_TYPE_SUB_SYSTEM_CONTROLLER
SPECIFIC_TYPE_TV
SPECIFIC_TYPE_GATEWAY
SPECIFIC_TYPE_REPEATER_SLAVE
SPECIFIC_TYPE_VIRTUAL_NODE

SPECIFIC_TYPE_SECURE_EXTENDER
SPECIFIC_TYPE_POWER_STRIP
SPECIFIC_TYPE_SATELLITE_RECEIVER
SPECIFIC_TYPE_SATELLITE_RECEIVER_V2
SPECIFIC_TYPE_SIREN
SPECIFIC_TYPE_GENERAL_APPLIANCE
SPECIFIC_TYPE_KITCHEN_APPLIANCE
SPECIFIC_TYPE_LAUNDRY_APPLIANCE
SPECIFIC_TYPE_NOTIFICATION_SENSOR
SPECIFIC_TYPE_DOORBELL
SPECIFIC_TYPE_SOUND_SWITCH
SPECIFIC_TYPE_SECURE_KEYPAD
SPECIFIC_TYPE_ZONED_SECURITY_PANEL

SPECIFIC_TYPE_SECURE_DOOR
SPECIFIC_TYPE_SECURE_GATE
SPECIFIC_TYPE_SECURE_BARRIER_ADDON
SPECIFIC_TYPE_SECURE_BARRIER_OPEN_ONLY
SPECIFIC_TYPE_SECURE_BARRIER_CLOSE_ONLY

Supported icon types:

Icon Type
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_WALL_LAMP
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_LAMP_POST_HIGH
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_LAMP_POST_LOW
ICON_TYPE_GENERIC_LIGHT_DIMMER_SWITCH
ICON_TYPE_SPECIFIC_LIGHT_DIMMER_SWITCH_PLUGIN
ICON_TYPE_SPECIFIC_LIGHT_DIMMER_SWITCH_WALL_OUTLET

ICON_TYPE_SPECIFIC_LIGHT_DIMMER_SWITCH_CEILING_OUTLET
ICON_TYPE_GENERIC_FAN_SWITCH
ICON_TYPE_GENERIC_DIMMER_WALL_SWITCH
ICON_TYPE_SPECIFIC_DIMMER_WALL_SWITCH_ONE_BUTTON
ICON_TYPE_SPECIFIC_DIMMER_WALL_SWITCH_TWO_BUTTONS
ICON_TYPE_SPECIFIC_DIMMER_WALL_SWITCH_THREE_BUTTONS
ICON_TYPE_SPECIFIC_DIMMER_WALL_SWITCH_FOUR_BUTTONS
ICON_TYPE_SPECIFIC_DIMMER_WALL_SWITCH_ONE_ROTARY
ICON_TYPE_GENERIC_ON_OFF_POWER_SWITCH
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_PLUGIN
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_WALL_OUTLET
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_CEILING_OUTLET
ICON_TYPE_GENERIC_ON_OFF_WALL_SWITCH

ICON_TYPE_SPECIFIC_ON_OFF_WALL_SWITCH_ONE_BUTTON
ICON_TYPE_SPECIFIC_ON_OFF_WALL_SWITCH_TWO_BUTTONS
ICON_TYPE_SPECIFIC_ON_OFF_WALL_SWITCH_THREE_BUTTONS
ICON_TYPE_SPECIFIC_ON_OFF_WALL_SWITCH_FOUR_BUTTONS
ICON_TYPE_SPECIFIC_ON_OFF_WALL_SWITCH_ONE_ROTARY
ICON_TYPE_GENERIC_VALVE_OPEN_CLOSE
ICON_TYPE_GENERIC_IRRIGATION
ICON_TYPE_SPECIFIC_LIGHT_DIMMER_SWITCH_DIN_RAIL_MODULE
ICON_TYPE_SPECIFIC_ON_OFF_POWER_SWITCH_DIN_RAIL_MODULE
ICON_TYPE_GENERIC_THERMOSTAT
ICON_TYPE_GENERIC_DOOR_LOCK_KEYPAD
ICON_TYPE_GENERIC_ENTRY_CONTROL
ICON_TYPE_SPECIFIC_ENTRY_CONTROL_RFID_TAG_READER_NO_BUTTON

ICON_TYPE_GENERIC_WINDOW_COVERING_NO_POSITION_ENDPOINT
ICON_TYPE_GENERIC_WINDOW_COVERING_ENDPOINT_AWARE
ICON_TYPE_GENERIC_WINDOW_COVERING_POSITION_ENDPOINT_AWARE
ICON_TYPE_GENERIC_REMOTE_CONTROL_AV
ICON_TYPE_GENERIC_REMOTE_CONTROL_MULTI_PURPOSE
ICON_TYPE_GENERIC_REMOTE_CONTROL_SIMPLE
ICON_TYPE_SPECIFIC_REMOTE_CONTROL_SIMPLE_KEYFOB
ICON_TYPE_GENERIC_WALL_CONTROLLER
ICON_TYPE_GENERIC_CENTRAL_CONTROLLER
ICON_TYPE_GENERIC_DISPLAY_SIMPLE
ICON_TYPE_GENERIC_GATEWAY
ICON_TYPE_GENERIC_SET_TOP_BOX
ICON_TYPE_GENERIC_SUB_SYSTEM_CONTROLLER

ICON_TYPE_GENERIC_TV
ICON_TYPE_GENERIC_REPEATER
ICON_TYPE_SPECIFIC_REPEATER_SLAVE
ICON_TYPE_SPECIFIC_IR_REPEATER
ICON_TYPE_GENERIC_POWER_STRIP
ICON_TYPE_GENERIC_SIREN
ICON_TYPE_GENERIC_SOUND_SWITCH
ICON_TYPE_SPECIFIC_SOUND_SWITCH_CHIME
ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_PEST_CONTROL
ICON_TYPE_SPECIFIC_ON_OFF_WALL_SWITCH_DOOR_BELL
ICON_TYPE_SPECIFIC_SOUND_SWITCH_DOORBELL
ICON_TYPE_SPECIFIC_ENTRY_CONTROL_KEYPAD_0_9
ICON_TYPE_SPECIFIC_ENTRY_CONTROL_KEYPAD_0_9_OK_CANCEL

ICON_TYPE_SPECIFIC_ENTRY_CONTROL_KEYPAD_0_9_OK_CANCEL_HOME_STAY_AWAY
ICON_TYPE_GENERIC_BARRIER

Entry lock schedules

Types of entry lock schedule

Week day entry lock schedule

```
{
    weekDay = "monday",
    startHour = 11,
    startMinute = 20,
    stopHour = 13,
    stopMinute = 50
}
```

field	type	description
weekDay	string	select one of week days
startHour	int	
startMinute	int	
stopHour	int	
stopMinute	int	

Year day entry lock schedule

```
{
    startYear = 2019,
    startMonth = 3,
    startDay = 1,
    startHour = 18,
    startMinute = 30,
    stopYear = 2019,
    stopMonth = 3,
    stopDay = 2,
    stopHour = 18,
    stopMinute = 30,
}
```

field	type	description
startYear	int	
startMonth	int	from 1 to 12
startDay	int	from 1
startHour	int	from 0 to 23
startMinute	int	from 0 to 59
stopYear	int	
stopMonth	int	from 1 to 12

stopDay	int	from 1
stopHour	int	from 0 to 23
stopMinute	int	from 0 to 59

Daily repeating entry lock schedule

```
{
    weekDays = { "monday", "friday" },
    startHour = 11,
    startMinute = 20,
    durationHour = 1,
    durationMinute = 50
}
```

field	type	description
weekDays	array of strings	select from week days
startHour	int	
startMinute	int	
durationHour	int	
durationMinute	int	

Command classes

Default

Default command class structure

```
{
    id = 50,
    version = 1,
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class

Associations

```
{
    id = 50,
    version = 1,
    associationGroupsCount = 2
    associationGroups = {
        {groupId = 1, groupNodes = [1, 2, 3]},
        {groupId = 2, groupNodes = [1, 5, 6]},
        ...
    }
}
```

field	type	description
id	int	identifier of command class

version	int	version of command class
associationGroupsCount	int	number of association groups
associationGroups	table	info about each association group

Association group info

```
{
    id = 50,
    version = 1,
    associationGroupsCount = 2
    associationGroups = {
        {
            groupId = 1,
            groupName = "Lifeline",
            profileMsb = 0,
            profileLsb = 0,
            commands = [{ccId = 71, commandId = 1}]
        },
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
associationGroupsCount	int	number of association groups

associationGroups	table	info about each association group

Binary sensor

```
{
    id = 50,
    version = 1,
    sensorTypes = {
        1,
        2,
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
sensorTypes	table	array of supported sensors

Central scene

```
{
    id = 91,
    version = 1,
    buttons = {
        {buttonId = 1, buttonAttributes={"press_1_time", "press_2_times"}},
        {buttonId = 2, buttonAttributes={"held_down", "released"}},
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
buttons	table	array of supported buttons

Color Switch

```
{
    id = 51,
    version = 1,
    componentIds = {
        wwhite,
        red,
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
componentIds	table	array of supported color ids

Meter

```
{
    id = 50,
```

```
    version = 1,
    reset = 1,
    type = 1,
    scales = {
        1,
        2,
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
reset	int	
type	type	type of metering physical unit
scales	table	array of measurement units

Multi channel associations

```
{
    id = 50,
    version = 1,
    associationGroupsCount = 2,
    associationGroups = {
        {groupId = 1, groupNodes = [{nodeId = 1, endpointId = 0}, {nodeId = 2,
endpointId = 0}]},
        {groupId = 2, groupNodes = [{nodeId = 1, endpointId = 0}, {nodeId = 3,
endpointId = 0}]},
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
associationGroupsCount	int	number of association groups
associationGroups	table	info about each association group

Sound switch

```
{
    id = 50,
    version = 1,
    tones = {
        { toneId = 1, duration = 5,  toneName = "01 Ding Dong" },
        { toneId = 2, duration = 9,  toneName = "02 Ding Dong Tubular" },
        { toneId = 3, duration = 10, toneName = "03 Traditional Apartment Buzzer" },
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class

tones	table	array of supported tones

Thermostat mode

```
{
    id = 64,
    version = 1,
    modes = {
        0,
        1,
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
modes	table	array of supported modes

Thermostat fan mode

```
{
    id = 68,
    version = 1,
    modes = {
        0,
        1,
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
modes	table	array of supported modes

Thermostat setpoint

```
{
    id = 67,
    version = 1,
    setpoints = {
        {
            type = 1,
            minValue = 2,
            minScale = 0,
            maxValue = 35,
            maxScale = 0
        },
        ...
    }
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class

setpoints	table	array of supported setpoints
type	int	type of setpoint
minValue	int	minimum value for setpoint type
minScale	int	temperature scale of `minValue`
maxValue	int	maximum value for setpoint type
maxScale	int	temperature scale of `maxValue`

User code

```
{
    id = 99,
    version = 1,
    supportedUsersCount = 25
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
supportedUsersCount	int	number of supported users (from 1 to N)

Schedule entry lock

```
{
    id = 78,
    version = 1,
    weekDaySlots = 0,
    yearDaySlots = 3,
    dailyRepeatingSlots = 7
}
```

field	type	description
id	int	identifier of command class
version	int	version of command class
weekDaySlots	int	number of supported week day slots
yearDaySlots	int	number of supported year day slots
dailyRepeatingSlots	int	number of supported daily repeating slots