python-openzwave Documentation

Release 0.4.0.35

bibi21000

Jun 14, 2017

Contents

Contents:

CHAPTER 1

libopenzwave module

CHAPTER 2

Data documentation

The common data structures and defitions.

CHAPTER 3

API documentation

Contents:

Network documentation

This is the central point. Everything is attached to a network.

Controller documentation

The controller is the node of your adaptater. You can use it to retrieve informations on it : library, statistics, ...

Option documentation

The options to start the manager. You can change the loglvel,...

Node documentation

The node.

This file is part of python-openzwave project https://github.com/OpenZWave/python-openzwave.

platform Unix, Windows, MacOS X

sinopsis openzwave API

License : GPL(v3)

class openzwave.node.**ZWaveNode**(*node_id*, *network*)
Represents a single Node within the Z-Wave Network.

add_value(*value_id*)
Add a value to the node

Parameters

- **value_id** (*int*) – The id of the value to add
- **command_class** (*str*) – The command_class of the value

Return type bool

assign_return_route()
Ask the to update its update its Return Route to the Controller

This command will ask a Node to update its Return Route to the Controller

Results of the AssignReturnRoute Command will be send as a Notification with the Notification type as Notification::Type_ControllerCommand

Returns True if the request was sent successfully.

Return type bool

basic
The basic type of the node.

Return type int

capabilities
The capabilities of the node.

Return type set()

change_value(*value_id*)
Change a value of the node. Not implemented

Parameters **value_id** (*int*) – The id of the value to change

command_classes
The commandClasses of the node.

Return type set()

command_classes_as_string
Return the command classes of the node as string.

Return type set()

create_button(*buttonid*)
Create a handheld button id.

Only intended for Bridge Firmware Controllers.

Results of the CreateButton Command will be send as a Notification with the Notification type as Notification::Type_ControllerCommand

Parameters **buttonid** (*int*) – the ID of the Button to query.

Returns True if the request was sent successfully.

Return type bool

delete_button(*buttonid*)
Delete a handheld button id.

Only intended for Bridge Firmware Controllers.

Results of the CreateButton Command will be send as a Notification with the Notification type as Notification::Type_ControllerCommand

Parameters **buttonid** (*int*) – the ID of the Button to query.

Returns True if the request was sent successfully.

Return type bool

device_type
The device_type of the node.

Return type str

generic
The generic type of the node.

Return type int

get_command_class_as_string(*class_id*)
Return the command class representation as string.

Parameters **class_id** (*hexadecimal code*) – the COMMAND_CLASS to get string representation

Return type str

get_command_class_genres()
Return the list of genres of command classes

Return type set()

get_max_associations(*groupidx*)
Gets the maximum number of associations for a group.

Parameters **groupidx** (*int*) – The group to query

Return type int

get_values(*class_id='All'*, *genre='All'*, *type='All'*, *readonly='All'*, *writeonly='All'*, *index='All'*, *label='All'*)
Retrieve the set of values. You can optionnaly filter for a command class, a genre and/or a type. You can also filter readonly and writeonly params.

This method always filter the values. If you wan't to get all the node's values, use self.values instead.

Parameters

- **class_id** (*hexadecimal code or string*) – the COMMAND_CLASS to get values
- **genre** (*'All' or PyGenres*) – the genre of value
- **type** (*'All' or PyValueTypes*) – the type of value

- **readonly** (*'All' or True or False*) – Is this value readonly
- **writeonly** (*'All' or True or False*) – Is this value writeonly
- **index** (*int*) – Index of value within all the values
- **label** (*str*) – Label of the value as set by openzwave

Return type set() of Values

get_values_by_command_classes(*genre='All'*, *type='All'*, *readonly='All'*, *writeonly='All'*)
Retrieve values in a dict() of dicts(). The dict is indexed on the COMMAND_CLASS. This allows to browse values grouped by the COMMAND_CLASS.You can optionnaly filter for a command class, a genre and/or a type. You can also filter readonly and writeonly params.

This method always filter the values. If you wan't to get all the node's values, use the property self.values instead.

Parameters

- **genre** (*'All' or PyGenres*) – the genre of value
- **type** (*'All' or PyValueTypes*) – the type of value
- **readonly** (*'All' or True or False*) – Is this value readonly
- **writeonly** (*'All' or True or False*) – Is this value writeonly

Return type dict(command_class : dict(valueids))

get_values_for_command_class(*class_id*)
Retrieve the set of values for a command class. Deprecated For backward compatibility only. Use get_values instead

Parameters **class_id** (*hexadecimal code or string*) – the COMMAND_CLASS to get values

Return type set() of classId

groups
Get the association groups reported by this node

In Z-Wave, groups are numbered starting from one. For example, if a call to GetNumGroups returns 4, the _groupIdx value to use in calls to GetAssociations AddAssociation and RemoveAssociation will be a number between 1 and 4.

Return type dict()

groups_to_dict(*extras=['all']*)
Return a dict representation of the groups.

Parameters **extras** (*[]*) – The extra inforamtions to add

Returns A dict

Return type dict()

has_command_class(*class_id*)
Check that this node use this commandClass.

Parameters **classId** (*hexadecimal code*) – the COMMAND_CLASS to check

Return type bool

heal(*upNodeRoute=False*)
Heal network node by requesting the node rediscover their neighbors. Sends a ControllerCommand_RequestNodeNeighborUpdate to the node.

Parameters **upNodeRoute** (*bool*) – Optional Whether to perform return routes initialization. (default = false).

Returns True is the ControllerCommand is sent. False otherwise

Return type bool

is_awake
Is this node a awake.

Return type bool

is_beaming_device
Is this node a beaming device.

Return type bool

is_failed
Is this node is presume failed.

Return type bool

is_frequent_listening_device
Is this node a frequent listening device.

Return type bool

is_info_received
Get whether the node information has been received. Returns True if the node information has been received yet

Return type bool

is_listening_device
Is this node a listening device.

Return type bool

is_locked
Is this node locked.

Return type bool

is_ready
Get whether the node is ready to operate (QueryStage Completed).

Return type bool

is_routing_device
Is this node a routing device.

Return type bool

is_security_device
Is this node a security device.

Return type bool

is_sleeping
Is this node sleeping.

Return type bool

is_zwave_plus
Is this node a zwave plus one.

Return type bool

location
The location of the node.

Return type str

manufacturer_id
The manufacturer id of the node.

Return type str

manufacturer_name
The manufacturer name of the node.

Return type str

max_baud_rate
Get the maximum baud rate of a node

name
The name of the node.

Return type str

neighbor_update()
Ask a Node to update its Neighbor Tables

This command will ask a Node to update its Neighbor Tables.

Results of the RequestNodeNeighborUpdate Command will be send as a Notification with the Notification type as Notification::Type_ControllerCommand

Returns True if the request was sent successfully.

Return type bool

neighbors
The neighbors of the node.

Return type set()

network_update()
Update the controller with network information from the SUC/SIS.

Results of the RequestNetworkUpdate Command will be send as a Notification with the Notification type as Notification::Type_ControllerCommand

Returns True if the request was sent successfully.

Return type bool

node_id
The id of the node.

Return type int

num_groups
Gets the number of association groups reported by this node.

Return type int

product_id
The product Id of the node.

Return type str

product_name
The product name of the node.

Return type str

product_type
The product type of the node.

Return type str

query_stage
Is this node a awake.

Return type string

refresh_info()
Trigger the fetching of fixed data about a node.

Causes the nodes data to be obtained from the Z-Wave network in the same way as if it had just been added. This method would normally be called automatically by OpenZWave, but if you know that a node has been changed, calling this method will force a refresh of the data held by the library. This can be especially useful for devices that were asleep when the application was first run.

Return type bool

refresh_value(*value_id*)
Refresh a value of the node. Not implemented

Parameters **value_id** (*int*) – The id of the value to change

remove_value(*value_id*)
Change a value of the node. Todo

Parameters **value_id** (*int*) – The id of the value to change

Returns The result of the operation

Return type bool

request_all_config_params()
Request the values of all known configurable parameters from a device.

request_config_param(*param*)
Request the value of a configurable parameter from a device.

Some devices have various parameters that can be configured to control the device behaviour. These are not reported by the device over the Z-Wave network but can usually be found in the devices user manual. This method requests the value of a parameter from the device, and then returns immediately, without waiting for a response. If the parameter index is valid for this device, and the device is awake, the value will eventually be reported via a ValueChanged notification callback. The ValueID reported in the callback will have an index set the same as _param and a command class set to the same value as returned by a call to Configuration::StaticGetCommandClassId.

Parameters **param** – The param of the node.

request_state()
Trigger the fetching of just the dynamic value data for a node. Causes the node's values to be requested from the Z-Wave network. This is the same as the query state starting from the dynamic state.

Return type bool

role
The role of the node.

Return type str

security
The security type of the node.

Returns The security type of the node

Return type int

send_information()
Send a NIF frame from the Controller to a Node. This command send a NIF frame from the Controller to a Node

Results of the SendNodeInformation Command will be send as a Notification with the Notification type as Notification::Type_ControllerCommand

Returns True if the request was sent successfully.

Return type bool

set_config_param(*param*, *value*, *size=2*)
Set the value of a configurable parameter in a device.

Some devices have various parameters that can be configured to control the device behaviour. These are not reported by the device over the Z-Wave network but can usually be found in the devices user manual. This method returns immediately, without waiting for confirmation from the device that the change has been made.

Parameters

- **param** – The param of the node.
- **value** – The value of the param.
- **size** (*int*) – Is an optional number of bytes to be sent for the parameter value. Defaults to 2.

Returns

Return type bool

set_field(*field*, *value*)
A helper to set a writable field : name, location, product_name, ...

Parameters

- **field** (*str*) – The field to set : name, location, product_name, manufacturer_name
- **value** (*str*) – The value to set

Return type bool

specific
The specific type of the node.

Returns The specific type of the node

Return type int

test(*count=1*)
Send a number of test messages to node and record results.

Parameters **count** (*int*) – The number of test messages to send.

to_dict(*extras=['all']*)
Return a dict representation of the node.

Parameters **extras** (*[]*) – The extra inforamtions to add

Returns A dict

Return type dict()

type
Get a human-readable label describing the node :rtype: str

values_to_dict (*extras=['all']*)
Return a dict representation of the values.

Parameters **extras** (*[ ]*) – The extra inforamtions to add

Returns A dict

Return type dict()

version
The version of the node.

Returns The version of the node

Return type int

Command documentation

The commands to use with nodes.

This file is part of python-openzwave project https://github.com/OpenZWave/python-openzwave.

platform Unix, Windows, MacOS X

sinopsis openzwave wrapper

License : GPL(v3)

python-openzwave is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

python-openzwave is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with python-openzwave. If not, see http://www.gnu.org/licenses.

class openzwave.command.**ZWaveNodeBasic**
Represents an interface to BasicCommands I known it's not necessary as they can be included in the node directly. But it's a good starting point.

What I want to do is provide an automatic mapping system hidding the mapping classes.

First example, the battery level, it's not a basic command but don't care. Its command class is 0x80.

A user should write

```
if self.handle_command_class(class_id):
    ret=command_Class(...)
```

The classic way to do it is a classic method of registering. But

Another way : using heritage multiple

ZWaveNode(ZWaveObject, ZWaveNodeBasic, ...) The interface will implement methods command_class_0x80(param1,param2,...) That's the first thing to do We also can define a property with a friendly name

handle_command_class will do the rest

Another way to do it : A node can manage actuators (switch, dimmer, ...) and sensors (temperature, consummation, temperature)

So we need a kind of mechanism to retrieve commands in a user friendly way Same for sensors.

A good use case is the AN158 Plug-in Meter Appliance Module We will study the following command classes : 'COMMAND_CLASS_SWITCH_ALL', 'COMMAND_CLASS_SWITCH_BINARY', 'COMMAND_CLASS_METER',

The associated values are :

```
COMMAND_CLASS_SWITCH_ALL : {
    72057594101481476L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': 'On and Off Enabled',
        'min': 0L,
        'writeonly': False,
        'label': 'Switch All',
        'readonly': False,
        'data_str': 'On and Off Enabled',
        'type': 'List'}
}
COMMAND_CLASS_SWITCH_BINARY : {
    72057594093060096L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': False,
        'min': 0L,
        'writeonly': False,
        'label': 'Switch',
        'readonly': False,
        'data_str': False,
        'type': 'Bool'}
}
COMMAND_CLASS_METER : {
    72057594093273600L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': False,
        'min': 0L,
        'writeonly': False,
        'label': 'Exporting',
        'readonly': True,
        'data_str': False,
        'type': 'Bool'},
    72057594101662232L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': 'False',
        'min': 0L,
```

```
        'writeonly': True,
        'label': 'Reset',
        'readonly': False,
        'data_str': 'False',
        'type': 'Button'},
    72057594093273090L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': 'kWh',
        'data': 0.0,
        'min': 0L,
        'writeonly': False,
        'label': 'Energy',
        'readonly': True,
        'data_str': 0.0,
        'type': 'Decimal'},
    72057594093273218L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': 'W',
        'data': 0.0,
        'min': 0L,
        'writeonly': False,
        'label': 'Power',
        'readonly': True,
        'data_str': 0.0,
        'type': 'Decimal'}
}
```

Another example from an homePro dimmer (not configured in openzwave):

```
COMMAND_CLASS_SWITCH_MULTILEVEL : {
    72057594109853736L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': 'False',
        'min': 0L,
        'writeonly': True,
        'label': 'Dim',
        'readonly': False,
        'data_str': 'False',
        'type': 'Button'},
    72057594109853697L: {
        'help': '',
        'max': 255L,
        'is_polled': False,
        'units': '',
        'data': 69,
        'min': 0L,
        'writeonly': False,
        'label': 'Level',
        'readonly': False,
        'data_str': 69,
        'type': 'Byte'},
```

```
    72057594118242369L: {
        'help': '',
        'max': 255L,
        'is_polled': False,
        'units': '',
        'data': 0,
        'min': 0L,
        'writeonly': False,
        'label': 'Start Level',
        'readonly': False,
        'data_str': 0,
        'type': 'Byte'},
    72057594109853720L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': 'False',
        'min': 0L,
        'writeonly': True,
        'label': 'Bright',
        'readonly': False,
        'data_str': 'False',
        'type': 'Button'},
    72057594118242352L: {
        'help': '',
        'max': 0L,
        'is_polled': False,
        'units': '',
        'data': False,
        'min': 0L,
        'writeonly': False,
        'label': 'Ignore Start Level',
        'readonly': False,
        'data_str': False,
        'type': 'Bool'}
}
```

What about the conclusion :

The COMMAND_CLASS_SWITCH_ALL is defined with the same label and use a list as parameter. This should be a configuration parameter. Don't know what to do for this command class

The COMMAND_CLASS_SWITCH_BINARY use a bool as parameter while COMMAND_CLASS_SWITCH_MULTILEVEL use 2 buttons : Dim and Bright. Dim and Bright must be done in 2 steps : set the level and activate the button.

So we must add one or more lines in the actuators :

Switch : {setter:self.set_command_class_0xYZ(valueId, new), getter:} We must find a way to access the value directly

Bright Dim

So for the COMMAND_CLASS_SWITCH_BINARY we must define a function called Switch (=the label of the value). What happen if we have 2 switches on the node : 2 values I suppose.

COMMAND_CLASS_SWITCH_MULTILEVEL uses 2 commands : 4 when 2 dimmers on the done ? Don't know but it can.

COMMAND_CLASS_METER export many values : 2 of them sends a decimal and are readonly. They also have a Unit defined ans values are readonly

COMMAND_CLASS_METER are used for sensors only. So we would map every values entries as defined before

Programming : get_switches : retrieve the list of switches on the node is_switch (label) : says if the value with label=label is a switch get_switch (label) : retrieve the value where label=label

can_wake_up()
Check if node contain the command class 0x84 (COMMAND_CLASS_WAKE_UP).

Filter rules are :

command_class = 0x84

Returns True if the node can wake up

Return type bool

get_battery_level(*value_id=None*)
The battery level of this node. The command 0x80 (COMMAND_CLASS_BATTERY) of this node.

Parameters **value_id** (*int*) – The value to retrieve state. If None, retrieve the first value

Returns The level of this battery

Return type int

get_battery_levels()
The command 0x80 (COMMAND_CLASS_BATTERY) of this node. Retrieve the list of values to consider as batteries. Filter rules are :

command_class = 0x80 genre = "User" type = "Byte" readonly = True writeonly = False

Returns The list of switches on this node

Return type dict()

get_config(*value_id=None*)
The command 0x70 (COMMAND_CLASS_CONFIGURATION) of this node. Set config to value (using value value_id)

Parameters **value_id** (*int*) – The value to retrieve value. If None, retrieve the first value

Returns The level of this battery

Return type int

get_configs(*readonly='All'*, *writeonly='All'*)
The command 0x70 (COMMAND_CLASS_CONFIGURATION) of this node. Retrieve the list of configuration parameters.

Filter rules are : command_class = 0x70 genre = "Config" readonly = "All" (default) or as passed in arg

Parameters

- **readonly** – whether to retrieve readonly configs
- **writeonly** – whether to retrieve writeonly configs

Returns The list of configuration parameters

Return type dict()

get_power_level(*value_id=None*)
The power level of this node. The command 0x73 (COMMAND_CLASS_POWERLEVEL) of this node.

Parameters **value_id** (*int*) – The value to retrieve state. If None, retrieve the first value

Returns The level of this battery

Return type int

get_power_levels()
The command 0x73 (COMMAND_CLASS_POWERLEVEL) of this node. Retrieve the list of values to consider as power_levels. Filter rules are :

command_class = 0x73 genre = "User" type = "Byte" readonly = True writeonly = False

Returns The list of switches on this node

Return type dict()

set_config(*value_id*, *value*)
The command 0x70 (COMMAND_CLASS_CONFIGURATION) of this node. Set config to value (using value value_id)

Parameters

- **value_id** (*int*) – The value to retrieve state
- **value** (*any*) – Appropriate value for given config

class openzwave.command.**ZWaveNodeSwitch**
Represents an interface to switches and dimmers Commands

get_dimmer_level(*value_id*)
The command 0x26 (COMMAND_CLASS_SWITCH_MULTILEVEL) of this node. Get the dimmer level (using value value_id).

Parameters **value_id** (*int*) – The value to retrieve level

Returns The level : a value between 0-99

Return type int

get_dimmers()
The command 0x26 (COMMAND_CLASS_SWITCH_MULTILEVEL) of this node. Retrieve the list of values to consider as dimmers. Filter rules are :

command_class = 0x26 genre = "User" type = "Bool" readonly = False writeonly = False

Returns The list of dimmers on this node

Return type dict()

get_rgbbulbs()
The command 0x33 (COMMAND_CLASS_COLOR) of this node. Retrieve the list of values to consider as RGBW bulbs. Filter rules are :

command_class = 0x33 genre = "User" type = "String" readonly = False writeonly = False

Returns The list of dimmers on this node

Return type dict()

get_rgbw(*value_id*)
The command 0x33 (COMMAND_CLASS_COLOR) of this node. Get the RGW value (using value value_id).

Parameters **value_id** (*int*) – The value to retrieve level

Returns The level : a value between 0-99

Return type int

get_switch_all_item(*value_id*)
The command 0x27 (COMMAND_CLASS_SWITCH_ALL) of this node. Return the current value (using value value_id) of a switch_all.

Parameters **value_id** (*int*) – The value to retrieve switch_all value

Returns The value of the value

Return type str

get_switch_all_items(*value_id*)
The command 0x27 (COMMAND_CLASS_SWITCH_ALL) of this node. Return the all the possible values (using value value_id) of a switch_all.

Parameters **value_id** (*int*) – The value to retrieve items list

Returns The value of the value

Return type set()

get_switch_all_state(*value_id*)
The command 0x27 (COMMAND_CLASS_SWITCH_ALL) of this node. Return the state (using value value_id) of a switch or a dimmer.

Parameters **value_id** (*int*) – The value to retrieve state

Returns The state of the value

Return type bool

get_switch_state(*value_id*)
The command 0x25 (COMMAND_CLASS_SWITCH_BINARY) of this node. Return the state (using value value_id) of a switch.

Parameters **value_id** (*int*) – The value to retrieve state

Returns The state of the value

Return type bool

get_switches()
The command 0x25 (COMMAND_CLASS_SWITCH_BINARY) of this node. Retrieve the list of values to consider as switches. Filter rules are :

command_class = 0x25 genre = “User” type = “Bool” readonly = False writeonly = False

Returns The list of switches on this node

Return type dict()

get_switches_all()
The command 0x27 (COMMAND_CLASS_SWITCH_ALL) of this node. Retrieve the list of values to consider as switches_all. Filter rules are :

command_class = 0x27 genre = “System” type = “List” readonly = False writeonly = False

Returns The list of switches on this node

Return type dict()

set_dimmer(*value_id*, *value*)
The command 0x26 (COMMAND_CLASS_SWITCH_MULTILEVEL) of this node. Set switch to value (using value value_id).

Parameters

- **value_id** (*int*) – The value to retrieve state
- **value** (*int*) – The level : a value between 0-99 or 255. 255 set the level to the last value. 0 turn the dimmer off

set_rgbw(*value_id*, *value*)
The command 0x33 (COMMAND_CLASS_COLOR) of this node. Set RGBW to value (using value value_id).

Parameters

- **value_id** (*String*) – The value to retrieve state
- **value** (*int*) – The level : a RGBW value

set_switch(*value_id*, *value*)
The command 0x25 (COMMAND_CLASS_SWITCH_BINARY) of this node. Set switch to value (using value value_id).

Parameters

- **value_id** (*int*) – The value to retrieve state
- **value** (*bool*) – True or False

set_switch_all(*value_id*, *value*)
The command 0x27 (COMMAND_CLASS_SWITCH_ALL) of this node. Set switches_all to value (using value value_id).

Parameters

- **value_id** (*int*) – The value to retrieve state
- **value** (*str*) – A predefined string

class openzwave.command.**ZWaveNodeSensor**
Represents an interface to Sensor Commands

get_sensor_value(*value_id*)
The command 0x30 (COMMAND_CLASS_SENSOR_BINARY) of this node. The command 0x31 (COMMAND_CLASS_SENSOR_MULTILEVEL) of this node. The command 0x32 (COMMAND_CLASS_METER) of this node.

Parameters **value_id** (*int*) – The value to retrieve value

Returns The state of the sensors

Return type variable

get_sensors(*type='All'*)
The command 0x30 (COMMAND_CLASS_SENSOR_BINARY) of this node. The command 0x31 (COMMAND_CLASS_SENSOR_MULTILEVEL) of this node. The command 0x32 (COMMAND_CLASS_METER) of this node. Retrieve the list of values to consider as sensors. Filter rules are :

command_class = 0x30-32 genre = "User" readonly = True writeonly = False

Parameters **type** (*'All' or PyValueTypes*) – the type of value

Returns The list of switches on this node

Return type dict()

class openzwave.command.**ZWaveNodeDoorLock**

Represents an interface to door lock and user codes associated with door locks

get_doorlock_logs()

The command 0x4c (COMMAND_CLASS_DOOR_LOCK_LOGGING) of this node. Retrieves the value consisting of log records. Filter rules are :

command_class = 0x4c genre = "User" type = "String" readonly = True

Returns The dict of log records with value_id as key

Return type dict()

get_doorlocks()

The command 0x62 (COMMAND_CLASS_DOOR_LOCK) of this node. Retrieves the list of values to consider as doorlocks. Filter rules are :

command_class = 0x62 genre = "User" type = "Bool" readonly = False writeonly = False

Returns The list of door locks on this node

Return type dict()

get_usercode(*index*)

Retrieve particular usercode value by index. Certain values such as user codes have index start from 0 to max number of usercode supported and is useful for getting usercodes by the index.

Parameters **index** (*int*) – The index of usercode value

Returns The user code at given index on this node

Return type *ZWaveValue*

get_usercodes(*index='All'*)

The command 0x63 (COMMAND_CLASS_USER_CODE) of this node. Retrieves the list of value to consider as usercodes. Filter rules are :

command_class = 0x63 genre = "User" type = "Raw" readonly = False writeonly = False

Returns The list of user codes on this node

Return type dict()

set_doorlock(*value_id*, *value*)

The command 0x62 (COMMAND_CLASS_DOOR_LOCK) of this node. Sets doorlock to value (using value_id).

Parameters

- **value_id** (*int*) – The value to retrieve state from
- **value** (*bool*) – True or False

set_usercode(*value_id*, *value*)
The command 0x63 (COMMAND_CLASS_USER_CODE) of this node. Sets usercode to value (using value_id).

Parameters

- **value_id** (*int*) – The value to retrieve state from
- **value** (*str*) – User Code as string

set_usercode_at_index(*index*, *value*)
The command 0x63 (COMMAND_CLASS_USER_CODE) of this node. Sets usercode to value (using index of value)

Parameters

- **index** (*int*) – The index of value to retrieve state from
- **value** (*str*) – User Code as string

Group documentation

The group is used in associations.

This file is part of python-openzwave project https://github.com/OpenZWave/python-openzwave.

platform Unix, Windows, MacOS X

sinopsis openzwave API

License : GPL(v3)

python-openzwave is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

python-openzwave is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with python-openzwave. If not, see http://www.gnu.org/licenses.

class openzwave.group.**ZWaveGroup**(*group_index*, *network=None*, *node_id=None*)
The driver object. Hold options of the manager Also used to retrieve information about the library, ...

add_association(*target_node_id*, *instance=0*)
Adds a node to an association group.

Due to the possibility of a device being asleep, the command is assumed to complete with success, and the association data held in this class is updated directly. This will be reverted by a future Association message from the device if the Z-Wave message actually failed to get through. Notification callbacks will be sent in both cases.

Parameters

- **target_node_id** (*int*) – Identifier for the node that will be added to the association group.
- **instance** (*int*) – The instance that will be added to the association group.

associations
The members of associations.

Return type set()

associations_instances
The members of associations with theirs instances. Nodes that does not support multi-instances have an instanceid equal to 0.

Return type set() of tuples (nodeid,instanceid)

index
The index of the group.

Return type int

label
The label of the group.

Return type int

max_associations
The number of associations.

Return type int

remove_association(*target_node_id*, *instance=0*)
Removes a node from an association group.

Due to the possibility of a device being asleep, the command is assumed to succeed, and the association data held in this class is updated directly. This will be reverted by a future Association message from the device if the Z-Wave message actually failed to get through. Notification callbacks will be sent in both cases.

Parameters

- **target_node_id** (*int*) – Identifier for the node that will be removed from the association group.
- **instance** (*int*) – The instance that will be added to the association group.

to_dict(*extras=['all']*)
Return a dict representation of the group.

Parameters **extras** (*[]*) – The extra inforamtions to add

Returns A dict

Return type dict()

Value documentation

The value.

This file is part of python-openzwave project https://github.com/OpenZWave/python-openzwave.

platform Unix, Windows, MacOS X

sinopsis openzwave API

License : GPL(v3)

python-openzwave is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

python-openzwave is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

General Public License for more details. You should have received a copy of the GNU General Public License along with python-openzwave. If not, see http://www.gnu.org/licenses.

class openzwave.value.**ZWaveValue**(*value_id*, *network=None*, *parent=None*)
Represents a single value.

check_data(*data*)
Check that data is correct for this value. Return the data in a correct type. None is data is incorrect.

Parameters **data** (*lambda*) – The data value to check

Returns A variable of the good type if the data is correct. None otherwise.

Return type variable

command_class
The command class of the value.

Returns The command class of this value

Return type int

data
Get the current data of the value.

Returns The data of the value

Return type depending of the type of the value

data_as_string
Get the value data as String.

Return type str

data_items
When type of value is list, data_items contains a list of valid values

Returns The valid values or a help string

Return type string or set

disable_poll()
Disable poll off this value.

Returns True if polling was disabled.

Return type bool

enable_poll(*intensity=1*)
Enable the polling of a device's state.

Parameters **intensity** (*int*) – The intensity of the poll

Returns True if polling was enabled.

Return type bool

genre
Get the genre of the value. The genre classifies a value to enable low-level system or configuration parameters to be filtered out by the application

Returns genre of the value (Basic, User, Config, System)

Return type str

help
Gets a help string describing the value's purpose and usage.

Return type str

id_on_network
Get an unique id for this value.

The scenes use this to retrieve values

```
<Scene id="1" label="scene1">
        <Value homeId="0x014d0ef5" nodeId="2" genre="user" commandClassId="38
↪" instance="1" index="0" type="byte">54</Value>
</Scene>
```

The format is :

home_id.node_id.command_class.instance.index

index
Get the value index. The index is used to identify one of multiple values created and managed by a command class. In the case of configurable parameters (handled by the configuration command class), the index is the same as the parameter ID.

Returns index of the value

Return type int

instance
Get the command class instance of this value. It is possible for there to be multiple instances of a command class, although currently it appears that only the SensorMultilevel command class ever does this.

Returns instance of the value

Return type int

is_change_verified()
determine if value changes upon a refresh should be verified. If so, the library will immediately refresh the value a second time whenever a change is observed. This helps to filter out spurious data reported occasionally by some devices.

is_polled
Verify that the value is polled.

Return type bool

is_read_only
Test whether the value is read-only.

Returns True if the value cannot be changed by the user.

Return type bool

is_set
Test whether the value has been set.

Returns True if the value has actually been set by a status message from the device, rather than simply being the default.

Return type bool

is_write_only
Test whether the value is write-only.

Returns True if the value can only be written to and not read.

Return type bool

label
Get the label of the value.

Return type str

max
Gets the maximum that this value may contain.

Return type int

min
Gets the minimum that this value may contain.

Return type int

node
The value_id of the value.

parent_id
Get the parent_id of the value.

poll_intensity
The poll intensity of the value.

Returns 0=none, 1=every time through the list, 2-every other time, etc

Return type int

precision
Gets a float value's precision.

Returns a float value's precision

Return type int

refresh()
Refresh the value.

Returns True if the command was transmitted to controller

Return type bool

set_change_verified(*verify*)
Sets a flag indicating whether value changes noted upon a refresh should be verified.

If so, the library will immediately refresh the value a second time whenever a change is observed. This helps to filter out spurious data reported occasionally by some devices.

Parameters **verify** (*bool*) – if true, verify changes; if false, don't verify changes.

to_dict(*extras=['all']*)
Return a dict representation of the node.

Parameters **extras** (*[]*) – The extra inforamtions to add

Returns A dict

Return type dict()

type
Get the type of the value. The type describes the data held by the value and enables the user to select the correct value accessor method in the Manager class.

Returns type of the value

Return type str

units
Gets the units that the value is measured in.

Return type str

value_id
Get the value_id of the value.

Scene documentation

The scenes.

This file is part of python-openzwave project https://github.com/OpenZWave/python-openzwave.

platform Unix, Windows, MacOS X

sinopsis openzwave API

License : GPL(v3)

python-openzwave is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

python-openzwave is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with python-openzwave. If not, see http://www.gnu.org/licenses.

class openzwave.scene.**ZWaveScene**(*scene_id*, *network=None*)
Represents a single scene within the Z-Wave Network

activate()
Activate the zwave scene.

Returns True if the scene is activated. False otherwise.

Return type bool

add_value(*value_id*, *value_data*)
Add a value with data value_data to the zwave scene.

Parameters

- **value_id** (*int*) – The id of the value to add
- **value_data** (*variable*) – The data of the value to add

create(*label=None*)
Create a new zwave scene on the network and update the object_id field If label is set, also change the label of the scene

Parameters **label** (*str or None*) – The new label

Returns return the id of scene on the network. Return 0 if fails

Return type int

get_values()
Get all the values of the scene

Returns A dict of values : {value_id={'value'=ZWaveValue, 'data'=data}, ...}.

Return type dict()

get_values_by_node()
Get all the values of the scene grouped by nodes

Returns A dict of values : {node_id={value_id={'value'=ZWaveValue, 'data'=data}, ...},...}.

Return type dict()

label
The label of the scene.

Return type str

remove_value(*value_id*)
Remove a value from the scene.

Parameters **value_id** (*int*) – The id of the value to change

Returns True if the scene is removed. False otherwise.

Return type bool

scene_id
The id of the scene.

Return type int

set_value(*value_id*, *value_data*)
Set a value data to value_data in the zwave scene.

Parameters

- **value_id** (*int*) – The id of the value to add
- **value_data** (*variable*) – The data of the value to add

to_dict(*extras=['kvals']*)
Return a dict representation of the node.

Parameters **extras** (*[]*) – The extra inforamtions to add

Returns A dict

Return type dict()

Object documentation

The low level object. Implements cache mechanism.

This file is part of python-openzwave project https://github.com/OpenZWave/python-openzwave.

platform Unix, Windows, MacOS X

sinopsis openzwave API

License : GPL(v3)

python-openzwave is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

python-openzwave is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with python-openzwave. If not, see http://www.gnu.org/licenses.

class openzwave.object.**ZWaveObject**(*object_id*, *network=None*, *use_cache=True*)
Represents a Zwave object. Values, nodes, ... can be changer by other managers on the network.

cache_property(*prop*)
Add this property to the cache manager.

Parameters **prop** (*lambda*) – The property to cache

home_id
The home_id of the node.

Return type int

is_outdated(*prop*)
Check if property information is outdated.

Parameters **prop** (*lambda*) – The property to check

Return type bool

kvals
The keyvals store in db for this object.

Return type {}

last_update
The last update date of the device.

Return type time

network
The network of the node.

Return type ZWaveNetwork

object_id
The id of the object. object_id could be None, when creating a scene for example.

Return type int

outdate(*prop*)
Says that the property information is outdated.

Parameters **prop** (*lambda*) – The property to outdate

outdated
Are the information of this object outdated.

How to manage the cache ?

2 ways of doing it : - refresh information when setting the property - refresh information when getting getting property. Maybe whe could implement the 2 methods.

Return type int

update(*prop*)
Says that the property are updated.

Parameters **prop** (*lambda*) – The property to update

use_cache
Should this object use cache from property

Return type bool

class openzwave.object.**ZWaveNodeInterface**
Represents an interface of a node. An interface can manage specific commandClasses (ie a switch, a dimmer, a thermostat, ...). Don't know what to do with it now but sure it must exist

exception openzwave.object.**ZWaveException**(*value*)
Exception class for OpenZWave

exception openzwave.object.**ZWaveTypeException**(*value*)
Exception class for OpenZWave

exception openzwave.object.**ZWaveCacheException**(*value*)
Exception class for OpenZWave

exception openzwave.object.**ZWaveCommandClassException**(*value*)
Exception class for OpenZWave

CHAPTER 4

Indices and tables

- genindex

Python Module Index

l

o

Index

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

Z