Z-Way Developers Documentation

(c) Z-Wave.Me Team, based on Version 1.4

Contents

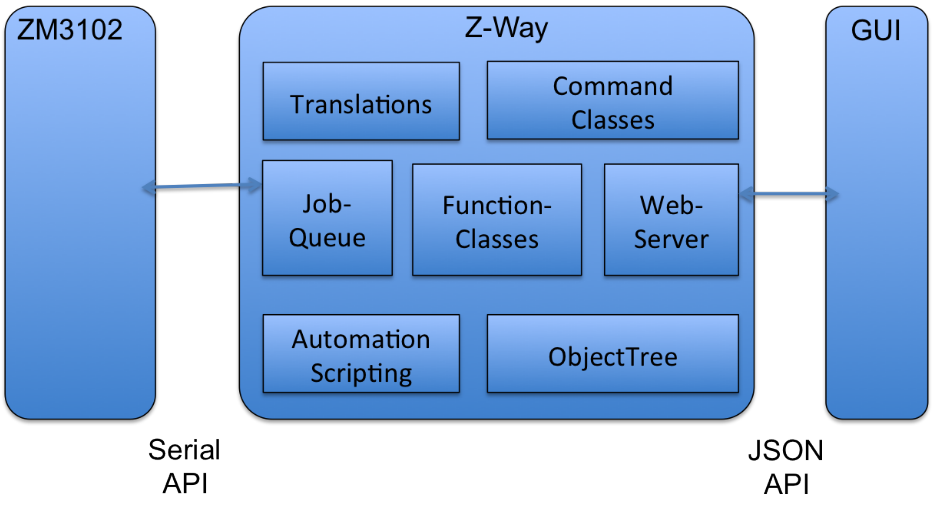


Figure 1: Z-Way Software Structure

1 The Z-Way software architecture

Z-Way is a fully featured home automation controller supporting Z-Wave as communication technology. It allows to

- Include and exclude devices and configure these devices, manage the network configuration and stability by visualizing the configuration and routing within the network
- Switch actuators such as electrical switches, dimmers, motor controls for sun blind, garage doors or venetian blind, door looks, heating thermostats and many more
- Access sensor data such as motion detection, temperature, CO2, smoke etc.
- Visualization of all functions of the Z-Wave network mapped to the floor plan or as tables simple to read
- Create logical connection between events created by sensors and actions performed by actuators

Z-Way communicates (south bound) to the Z-Wave transceiver firmware (using the Serial Interface) and offers a northbound interface that complies to the JSON specification (for details about JSON see explanation in the chapters below). This north bound Interface - referred to as JSON Interface - is used by applications or web pages (with Javascript) to operate and use the Z-Wave network. It is possible that multiple user interfaces or applications run in parallel and use the JSON API. However in case they send contradicting messages (one is turning off a device while one is turning on) the resulting state of the network is unpredictable.

Z-Way consists of several function blocks:

- The Job Queue: This is the core of Z-Way
- Function Classes: The implementation of all the commands to control the Z-Wave transceiver chip and the Z-Wave network

- Command Classes: The application level commands used to control Z-Wave devices in the network
- The JSON web server: It implements the application programmers interface
- Translation Functions: They help to translate machine readable tokens into human-readable strings
- The automation and scripting engine: This is the way to get the intelligence into the system.

2 How to use Z-Way and this manual

Z-Way comes with several user interfaces among them the Demo UI. The Demo UI is written in AJAX Technology and exposes all functions of Z-Way and the Z-Wave network controlled. This Demo UI will typically be the starting point for new users. The description of this Demo User Interface is not scope of this manual. Please refer to the Z-Way Users Manual for more information.

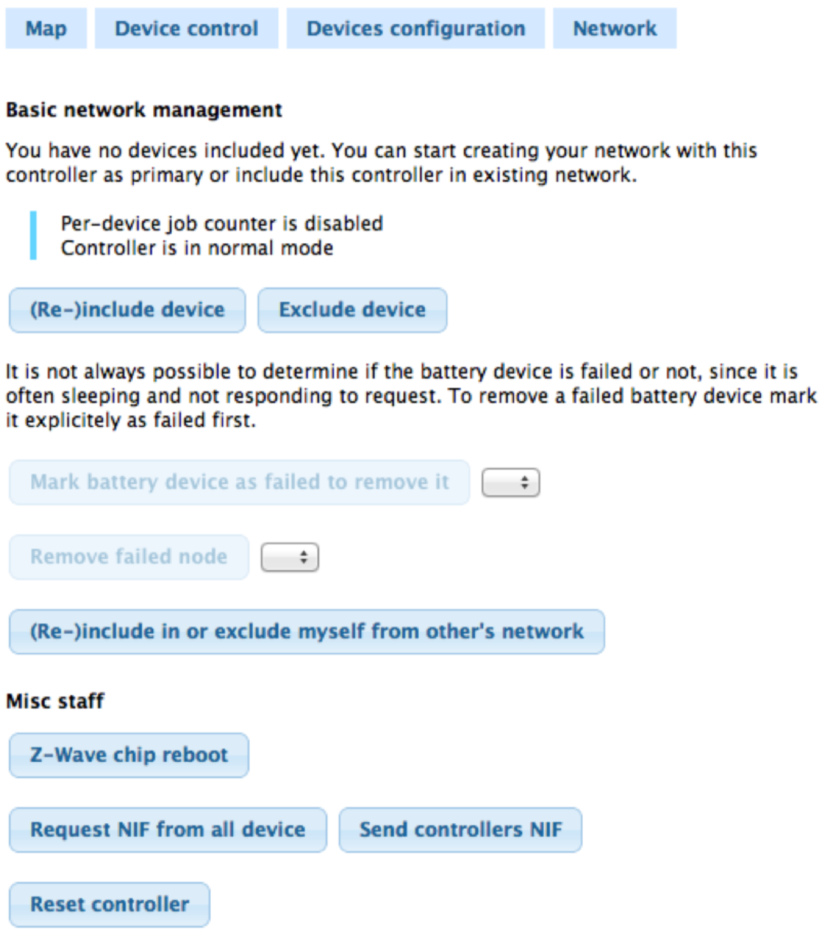
Before a device can be used the device needs to be included in the Z-Wave network managed by Z-Way. This management function is accessible in the Demo UI under the Tab 'Network'. Its one of the functions to manage the network. The chapter "Function Classes" explains te different management functions and how they can be used and will use the Demo UI dialogs as application example. The Chapter "Function Class Reference" documents all function classes available.

The control of devices is implemented in Command Classes. the chapter "Command Class implementation is again using certain dialogs of the Demo UI to explain how to access these functions. (The chapter Command Class Reference documents all command class functions).

In order to access device and network related data in Z-Way knowledge of the Z-Way data model is essential. The chapter "The Z-Way Data Model" gives the necessary insight into the data model. All data of Z-Way is exposed on the Demo UI.

There are three ways to access and work with Z-Way. All three Application Programmers Interfaces use the very same function and command classes and the same data model but differ in their syntax and they way they access data.

- JSON API This API is available on the TCP port configured. This interface is used by the Demo UI and most of the UIs. The chapter 'JSON-API" explains the use of this interface.
- Automation API The automation function is explained in the chapter 'Automation Engine'
- The C level API All functions of the JSON API are bound to C calls . The folder "devel" contains the C style header file needed to use this API.



Basic network management

You have no devices included yet. You can start creating your network with this controller as primary or include this controller in existing network.

Per-device job counter is disabled
Controller is in normal mode

It is not always possible to determine if the battery device is failed or not, since it is often sleeping and not responding to request. To remove a failed battery device mark it explicitely as failed first.

Mark battery device as failed to remove it
Remove failed node
(Re-)include in or exclude myself from other's network

Misc staff

Figure 2: Demo UI Dialog for Networking functions

3 Function Class Implementation

The commands controlling the Z-Wave transceiver firmware using the serial interface are called **Function Classes**. Most functions to manage the network are implemented in function classes. One special function class is 'SendData' because this is the class used to send application commands to the transceiver chip.

Most function classes are used by Z-Way alone to manage the Z-Wave network but some function classes are exposed to the user interface to enabled manual interaction with the Z-Wave network. The function classes exposed can be used on the demo user interface on the tab 'network'.

3.1 Inclusion

You can include devices by pressing the Include Device button. This turns the controller into an inclusion mode that allows including a device. A status information line indicates this status.

The inclusion of a device is typically confirmed with a triple press of a button of this particular device. However, please refer to the manual of this particular device for details how to include them into a Z-Wave network. The inclusion mode will time out after about 20 seconds or is aborted by pressing the Stop Include button.

If the network has a special controller with SIS function (Z-Way will try to activate such as function on default, hence this mode should always be active if the USB hardware used by Z-Way supports it) the inclusion of further devices can also be accomplished by using the include function of any portable remote control which is already included into the network. A short explanation above the include button will inform about the ways devices can be included.

The Inclusion function is implemented using the function class **AddNodeToNetwork(flag)** with flag=1 for starting the inclusion mode and flag =0 for stopping the inclusion mode. Please refer to the chapter 14 for details on how to use this function.

3.2 Exclusion

You can exclude devices by pressing the Exclude Device button. This turns the controller into an exclusion mode that allows excluding a device. The exclusion of a device is typically confirmed with a triple press of a button of this particular device as well. However, please refer to the manual of this device for details how to exclude them into a Z-Wave network. The exclusion mode will time out after about 20 seconds or is aborted by pressing the Stop Exclude button. It is possible to exclude all kind of devices regardless if they were included in the particular network of the excluding controller.

If a node is not longer in operation it cant be excluded from the network since exclusion needs some confirmation from the device. Please use the Remove Failed Node function in this case. Please make sure that only failed nodes are moved this way. Removed but still function nodes - called phantom nodes will harm the network stability.

The Exclusion function is implemented using the function class **RemoveNodeToNetwork(flag)** with flag=1 for starting the exclusion mode and flag =0 for stopping the exclusion mode. Please refer to the chapter 14 for details on how to use this function.

3.3 Mark Battery powered devices as failed

This function allows marking battery-powered devices as failed. Only devices marked as failed can be excluded from the network without using the exclusion function. Typically multiple failed communications with a device result in this marking. Battery powered devices are recognized as sleeping in the controller and therefore all communication attempts with this device will be queued until a wakeup notification from this device is received. A faulty battery operated device will never send a wakeup notification and hence there is never a communication, which would result in a failed node status. Battery operated devices can therefore be manually marked as faulty. Make sure to only mark - and subsequently remove devices that are faulty or have disappeared. A device, which was removed with this operation but is still functioning it may create malfunctions in the network.

This function is no Function class but sets the internal 'failed' valuable of the device.

3.4 Remove Failed Nodes

This function allows you to remove nodes, which are not longer responding or which are not available. Please refer to the manual section Network stability for further information about why failed nodes should be removed.

Z-Way allows removing a node, if and only if this node was detected as failed by the Z-Wave transceiver. The network will recognize that communication with a device fails multiple times and the device cant be reached using alternating routes either. The controller will then mark the device as failed but will keep it in the current network configuration. Any successful communication with the device will remove the failed mark. Only devices marked as failed can be removed using the Remove Failed Node function.

If you want to remove a node that is in operation use the Exclude Function.

This function is implemented using the function class **RemoveFailedNode(node id)** with node id as the node id of the device to be removed. Please refer to the chapter 14 for details on how to use this function. It is also possible to replace a failed node by a new node using the function class **RemoveFailedNode(node id)**. This function is not shown in the demo user interface but please refer to the chapter 14 for details.

The function **IsFailedNode(node id)** can be used to detect if a certain node is failed. The Z-wave transceiver will try to contact the device wirelessly and will then update the failed-status inside the transceiver and also the failed-status of the device in Z-Way.

3.5 Include into / Exclude from a different network

Z-Way can join a Z-Wave network as secondary controller. It will change its own Home ID to the Home ID of the new network and it will learn all network information from the including controller of the new network. To join a different network, the primary controller of this new network need to be in the inclusion mode.

Z-Way needs to be turned into the learn mode using the button Start Include in others network. The button Stop Include in others network can be used to turn off the Learn mode, which will time out otherwise or will stop if the learning was successful.

Please be aware that all existing relationships to existing nodes will get lost when the Z-Way controller joins a different network. Hence it is recommended to join a different network only after a reset with no other nodes already included.

The Learn function is implemented using the function class **SetLearnMode(flag)** with flag=1 for starting the learn mode and flag =0 for stopping the learn mode. Please refer to the chapter 14 for details on how to use this function.

3.6 Z-Wave chip reboot

This function will perform a soft restart of the firmware of the Z-Wave controller chip without deleting any network information or setting. It may be necessary to recover the chip from a freezing state. A typical situation of a required chip reboot is if the Z-Wave chip fails to come back from the inclusion or exclusion state.

The reboot function is implemented using the function class **SerialAPISoftReset()** Please refer to the chapter 14 for details on how to use this function.

3.7 Request NIF from all devices

This function will call the Node Information Frame from all devices in the network. This may be needed in case of a hardware change or when all devices where included with a portable USB stick such as e.g. Aeon Labs Z-Stick. Mains powered devices will return their NIF immediately, battery operated devices will respond after the next wakeup.

This function controls a Z-Way controller function that will send out a function class **RequestNodeInformation(node)** to all nodes in the network. The function can also be called for one single node only. Please refer to the chapter 14 for details on how to use this function.

3.8 Send controllers NIF

In certain network configurations it may be required to send out the Node Information Frame of the Z-Way controller. This is particularly useful for the use of some remote controls for scene activation. The manual of the remote control will refer to this requirement and give further information when and how to use this function.

This function is implemented with the function class **SerialAPIApplicationNodeInfo** with plenty of parameters. These parameters are partly set by Z-Way but particularly the Command classes supported (parameter "NIF") can be changed by editing the file defaults.xml. Please refer to the chapter 14 for details on how to use this function the chapter about the translation files on how to change defaults.xml

3.9 Reset Controller

The network configuration (assigned node IDs and the routing table and some other network management specific parameters) is stored in the Z-Wave transceiver chip and will therefore even survive a complete reinstallation of the Z-Way software.

The function Reset Controller erases all values stored in the Z-Wave chip and sent the chip back to factory defaults. This means that all network information will be lost without recovery option.

This function may create problems if there are still devices included in the network, which are not reset to factory default (excluded) before the controller is reset. These devices may continue to communicate with the controller regardless if their Node IDs are stored in the controller after reset. This can cause all kind of problems. Hence, please handle this function with extreme caution! Z-Wave-ME hardware does not have this problem anymore!

This function is implemented with the function class **SetDefault()**. Please refer to the chapter 14 for details on how to use this function.

3.10 Change Controller

The controller change function allows to handover the primary function to a different controller in the network. The function works like a normal inclusion function but will hand over the primary privilege to the new controller after inclusion. Z-Way will become a secondary controller of the network. This function may be needed during installation of larger networks based on remote controls only where Z-Way is solely used to do a convenient network setup and the primary function is finally handed over one of the remote controls.

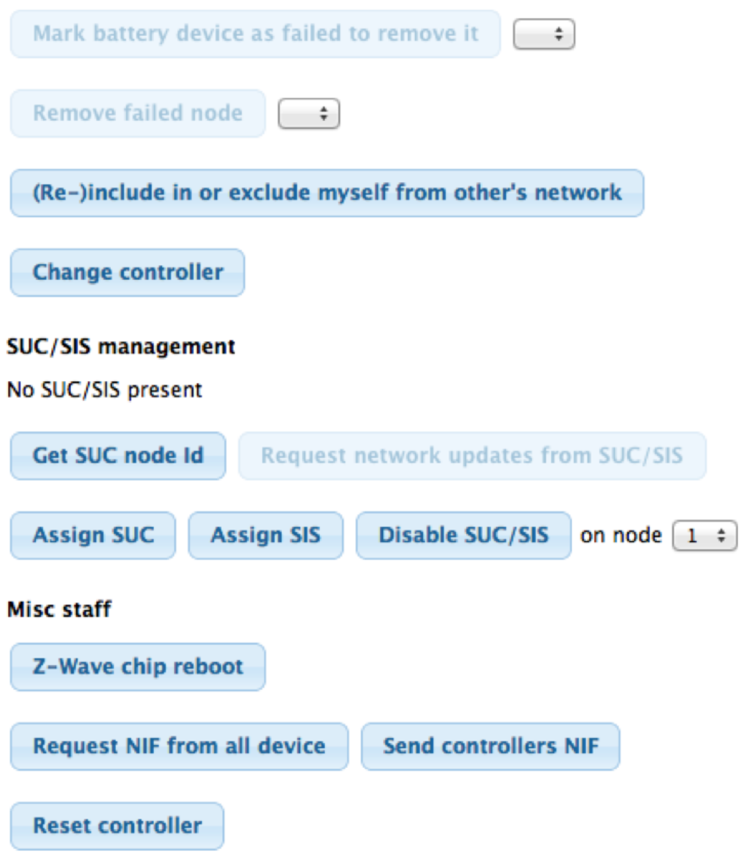
This function is implemented with the function class **ControllerChange**. Please refer to the chapter 14 for details on how to use this function.

3.11 SUC/SIS Management

This interface allows controlling the SUC/SIS function for the Z-Wave network. All these functions are almost obsolete and only needed for certain enhanced configurations of the Z-Wave network. Unless you really know what you do - don't use these functions!

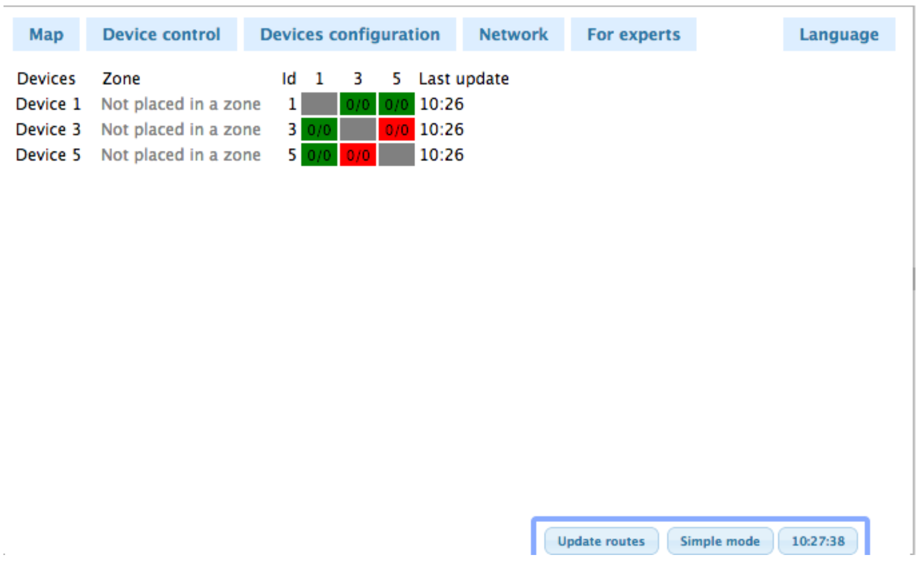
The following Function Classes are mapped to the demo user interface functions for SUC/SIS manipulation:

- GetSUCNodeId - get the SUC Node ID from the network
- EnableSUC - enables the SUC function in Z-Way, this is done by default if the transceiver firmware used supports SUC



No SUC/SIS present

Figure 3: Demo UI Dialog for Networking functions - experts functions



Devices	Zone	Id	1	3	5	Last update
Device 1	Not placed in a zone	1	▨	🟩 0/0	🟩 0/0	10:26
Device 3	Not placed in a zone	3	🟩 0/0	▨	🟥 0/0	10:26
Device 5	Not placed in a zone	5	🟩 0/0	🟥 0/0	▨	10:26

Figure 4: Demo UI dialog for Routing Table

- SetSUCNodeId - assign SUC function to a node in the network that is capable of running there SUC function.
- SendSUCNodeId - inform a different node about the node ID of a SUC in the system

3.12 Routing Table

The routing table of the Z-Wave network is shown in the tab network as well. It indicates how two devices of the Z-Wave network can communicate with each other. If two devices are in direct range (they can communicate without the help of any other node) the cross point of the two devices in the table is marked as dark green. The color light green indicates that the two nodes are not in direct range but have more than one alternating routes with one node between. This is still considered as a stable connection. The yellow color indicates that there are less than two one-hop routes available between the two nodes. However there may be more routes but with more nodes between and therefore considered as less stable.

A red indicator shows that there are no good short connections between the two nodes. This does not mean that they are unable to communicate with each other but any route with more than 2 routers between Z-Way is considering as not reliable, even taking into account that Z-Wave supports routes with up to four devices between. Grey cells indicate the connection to the own Node ID.

The general rule of thumb is: The greener the better

The table lists all nodes on the y-axis and the neighborhood information on the x-axis. On the right hand side of the table a timestamp shows when the neighborhood information for a given node was reported.

In theory the table should be totally symmetric, however different times of the neighborhood detection may result in different neighborhood information of the two devices involved.

The neighbor information of the controller works with an exception. The Z-Wave implementation used in current Z-Wave transceiver does not allow requesting an update of the neighbor list for the controller itself. The neighborhood information displayed for the controller s therefore simply wrong.

Battery powered devices will report their neighbors when woken up and report their mains powered neighbor correctly. However mains powered devices will report battery-powered devices as neighbors only when routes are updated twice. This is less critical because battery powered devices cant be used as routers and are therefore not relevant for calculating route between two nodes anyway.

The context menu command Network Reorganization allows re-detecting all neighborhood information (battery powered devices will report after their next wakeup!) Please refer to the manual section Network Stability for further information about the use of this function.

The routing table is stored in the Z-Wave transceiver and can be read using the function class **GetRoutingTableLine(node id)** for a given node ID. The function **RequestNodeNeighbourUpdate(nodeid)** will cause a certain node ID to redirect its wireless neighbors. It makes sense to call the GetRoutingTable function right after successful callback of the RequestNodeNeighbourUpdate function.

3.13 Hidden function classes

Beside the function classes used to manage the network there are some further groups of function classes that are used by Z-Way for maintenance and debugging purposes.

- Functions to change EERPOM contents - this is for advanced debugging only: MemoryGetByte, MemoryGetBuffer, MemoryPutByte, MemoryPutBuffer
- Functions to control the watchdog function of the Z-Wave transceiver - they are used by Z-Way only: WatchDogStart, WatchDogStop
- Functions to send Application Level Commands (Command Class Commands): SendData, SendDataAbort
- Functions to retrieve functionality, timing values and version information from the transceiver-firmware - they are used by Z-Way during startup: GetSerialAPICapabilities, SerialAPISetTimeouts, SerialAPIGetInitData, GetHomeId, GetControllerCapabilities, GetVersion
- Functions to manipulate the Routing Table entries of different nodes in the network - direct control of the routing table is not recommended and handled by Z- Way: AssignReturnRoute, AssignSUCReturnRoute, DeleteReturnRoute, DeleteSUCReturnRoute, RequestNetworkUpdate, CreateNewPrimary

4 Command Class Implementation

Z-Wave device functions are controlled by command classes. A command class can be have one or multiple commands allowing the use of a certain function of the device. Command classes are organized by functions, e.g. the command class "Switch Binary" allows to switch a binary switch. Hence the number of supported command classes define the functionality and the value of a device.

Z-Wave defines quite a few command classes but a lot of them are not implemented in any device. Hence, Z-Way will not implement them either. Command Classes consist of two types of commands:

- commands for users, most of them are either "GET" commands asking for a value or status from a remote device, or they are "SET" commands setting a certain value and therefore causing a certain action.
- commands for configuration

The commands for configuration are used by Z-way to build the data model used and to manage the device itself but they are not made public for users. The command classes available for users are listed in chapter 13.

Command to Z-Wave devices take time to be executed. In case the device is awake (mains powered or FLIRS) the delay may be well below one second but for battery powered devices the controller has to wait for the next scheduled wakeup in order to send the command.

In case the command is calling a value update from e.g. a sensor the successful execution of the command only means that the request was accepted by the wireless device. In order to really update a value the device itself needs to send a wireless command back to the controller that need to be accepted by the controller.

Each data element that is available in a remote wireless device (e.g. a switch) is also stored in Z-Way. First of all there is the assumption that the data value of the remote device and the corresponding data value in Z-Way are identical.

Most commands of command classes are eighter a "GET" asking for an update of an value or they are a "SET" - setting a new value. Setting a new value may also cause an action on the remote side. As an example switching a binary switch means in Z-Wave command classes changing the switching state value wirelessly.

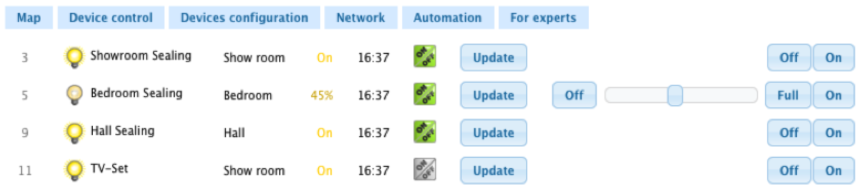
Z-Way will never "just" update the local corresponding value of the real remote value but always ask the remote device after a successful "SET" command for an update of the remote value using a "GET" command. This means that the local - displayable - value of a certain remote device will only update after some delay.

The process can be examined using the "Device Control" dialog in the demo user interface.

4.1 Switch Overview

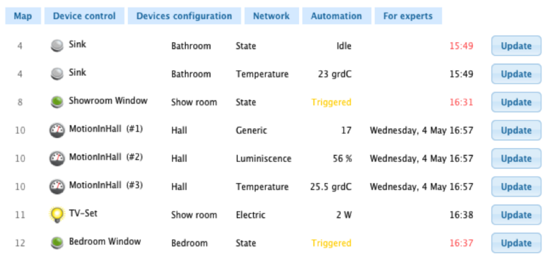
This page gives a table style overview of all actuators of the Z-Wave network. Actuators are devices with some kind of switching function such as

- Digital (on/off) switches,
- Light Dimmer,
- Motor Controls for Venetian blinds, window bind,
- Motor Control to open/close doors and windows.



3	Showroom Sealing	Show room	On	16:37		Update			Off On
5	Bedroom Sealing	Bedroom	45%	16:37		Update	Off		Full On
9	Hall Sealing	Hall	On	16:37		Update			Off On
11	TV-Set	Show room	On	16:37		Update			Off On

Figure 5: Demo UI Dialog for Switches



4	Sink	Bathroom	State	Idle		15:49	Update
4	Sink	Bathroom	Temperature	23 grdC		15:49	Update
8	Showroom Window	Show room	State	Triggered		16:31	Update
10	MotionInHall (#1)	Hall	Generic	17	Wednesday, 4 May	16:57	Update
10	MotionInHall (#2)	Hall	Luminiscence	56 %	Wednesday, 4 May	16:57	Update
10	MotionInHall (#3)	Hall	Temperature	25.5 grdC	Wednesday, 4 May	16:57	Update
11	TV-Set	Show room	Electric	2 W		16:38	Update
12	Bedroom Window	Bedroom	State	Triggered		16:37	Update

Figure 6: Demo UI Dialog for Sensors

Beside the name of the device, the location and the type of device the actual status and the timestamp of this status are shown.

Of course it is possible to switch the devices and to update the status of the device.

A little icon indicates how the device will react to a switch all devices command (will switch, will not switch, will react to off command only or to on command only).

The commands to control the different actors apply the command classes "Switch Multilevel" and "Switch Binary". Please refer to Chapter 13 for details how to use these command classes.

4.2 Sensor Overview

This page gives a table style overview of all sensors of the Z-Wave network. Sensors are devices able to report measured values. Sensors can report binary or analog values. Beside the name of the device, the location and the type of sensor the actual sensor value and the timestamp of this value are shown. It is possible to ask for an update of the sensor value.

The update command of the sensor interface and the values shown are using the command classes "Sensor Multilevel" and "SensorBinary". Please refer to Chapter 13 for details how to use these command classes.

4.3 Meter Overview

This page gives a table style overview of all meters of the Z-Wave network. Meters are devices able to report accumulated values. Beside the name of the device, the location and the type of meter the actual meter value and the timestamp of this value are shown. It is possible to ask for an update of the meter value.

The update command of the meter interface and the values shown are using the command class "Meter". Please refer to Chapter 13 for details how to use the meter command class.

4.4 Thermostat Overview

This page gives a table style overview of all thermostats of the Z-Wave network. Depending on the thermostat capabilities reported the dialog will allow to change the thermostat mode and/or change the setpoint temperature for the thermostat mode selected.

Please refer to Chapter 13 for details how to use the thermostat setpoint and thermostat mode command classes.

4.5 Door Lock Overview

This page gives a table style overview of all door locks of the Z-Wave network. Depending on the door lock reported the dialog will allow to open/close the door and differentiate on door handles.

Please refer to Chapter 13 for details how to use the door lock, user code and door lock loggingcommand classes.

4.6 Device Configuration

Beside the command class to directly control devices and update device values there are more command that are used to built the data model of the device and to get configuration values.

The devices also allow certain configurations itself. The demo UI shows the use of these command classes in the Tab "Device Configuration". The device configuration fulfills three basic tasks:

- The interview process after inclusion of the device
- The configuration of the device according to user requirements and needs.
- Setting Cause/effect relationships. This means that certain devices can directly control other devices.

4.6.1 Interview Process

After the inclusion of a new device Z-Way will interview this very device. The interview is a series of commands Z-Way is sending to the device in order to learn the capabilities and functions of this device.

Depending on the capabilities announced in the Node Information Frame that was received during the inclusion Z-Way will ask further questions to get more detailed information. The interview process may take some seconds since more questions may be required to ask depending in certain answers given. Since all functions of a device are grouped in so called Command Classes each command class announced in the Node Information frame will typically cause its part of the interview. The interview will be executes in three different Steps:

1. In case there is a Version Command Class ask for the Version of the device and the versions of all Command Classes announced in the Node Information frame. Otherwise Version 1 is assumed.
2. In case there is a Multi Channel Command class announced, ask for the number of the capabilities of the different channels and repeat Step 3 for each channel.
3. Ask for all capabilities of all command classes.
4. Do some auto configurations if needed.

The Device Status tab will indicate if the interview was successfully completed. The blue information icon shows if the interview was not complete. Clicking on this icon opens a dialog with all command classes and the status of their respective interviews. A complete interview is important in order to have access to all functions of the device included. Incomplete interviews may also be a reason for malfunctions of the network. There are several reasons why an interview may not be completed.

1. A battery-operated device may be gone into sleep mode too early. In this case its possible to wake up the device manually to complete the interview. Sometimes manual wakeup is needed several times.
2. The device does not fully comply with the Z-Wave protocol. This is particularly possible for devices that were brought to market before 2008. The current more sophisticated certification process makes sure that device are 100% compatible to the Z-Wave product when they hit the market. Please check online information (wiki, forums) on details and possible ways to fix these kinds of problems.
3. The device does not have a reliable communication route to the controller. Interview communication typically use longer packets than normal polling communication. This makes the interview communication more vulnerable against weak and instable communication links. Its possible that the controller is able to include a device and even receive confirmation of a polling request but still not being able to complete the interview. However this is a rare case.
4. The device may be simply broken.

Most of the command exchange during interview is using command class commands that are used during interview only and therefore are not exposed on the API.

4.6.2 Device Configuration

Each Z-Wave device is designed to work out of the box after inclusion without further configuration. However it may be suitable and in certain contexts even required to do a device specific configurations.

The device configuration page allows to further configure the device and to access certain additional information about the device. The tab is grouped into several sections. The sections can be toggled from invisible to visible and back by clicking on the headlines:

- Select Z-Wave Device Description Record
- Device Description
- Configurations
- Actions with configurations
- Advanced Actions

Select Z-Wave Device Description Record After a successful inclusion Z-Way will interview the device to gather further information.

Certain information such as names of association groups, the brand name of the device and the parameters of further configuration values can not be detected during interview. Z-Way uses a device database with product description files to obtain this information. In order to identify the right device description record certain parameters of the interview are used. If these parameters match exactly one device description record this very record is loaded and its content is shown on the device configuration page automatically.

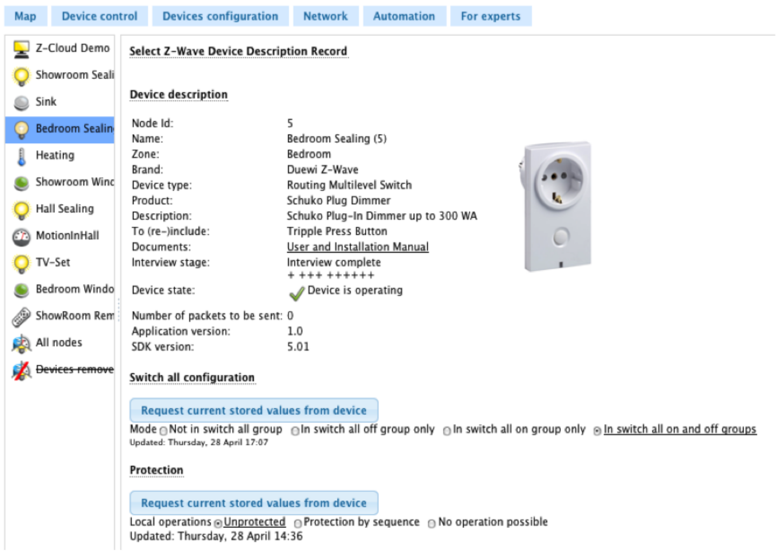
If the information from the device is sufficient to select one specific record from the database this section of the tab is hidden. If it is not possible to identify the correct device description record the user can manually choose the correct record. It is also possible to manually change the selection of the device description by unhiding this section and clicking on the Select Device Description Record button.

Device Description The upper part of the dialog shows some descriptive values of the device. The Z-Wave device type is the only value generated solely from the interview data. All other data are taken from the device description record.

- Zone: ... the zone/room the device is assigned to. Will be manually defined in Zone-tab.
- Brand: the product code or brand name of the device. This will be taken from the device description record.
- Device Type: ... the type of Z-Wave device as reported by the device during inclusion.
- Description: a verbal description of the function. This will be taken from the device description record.
- Interview Stage: shows the progress of the interview process. This information is generated by Z-Way.
- Inclusion Note: how to (re-) include the device. This will be taken from the device description record.
- Wakeup Note: this will be taken from the device description record.
- Documents: If the device description record offers links to manuals or other online documents there are shown here. This will be taken from the device description record.
- Device State: Status of the device plus number of packets queued for this device

There are a couple of reasons why no device description record was found:

1. There is no record for the device available. Since there are always new devices on the market Z-Way needs to catch up and update its device database. If your device is not found, updating to the most recent version of Z-Way may help.
2. The interview was not finished to the point where enough parameters were detected to identify the correct device description record. You may manually choose the correct device description record using the button Select Device Description Record. A dialog box will be opened for manual selection of the product (if available). The manual selection of a device description record is only needed if no record was found on default.
3. The interview of the device was completed but the device does not offer enough information to identify the correct device. You may manually choose the correct device description record using the button Select Device Description Record. A dialog box will be opened for manual selection of the product (if available). The manual selection of a device description record is only needed if no record was found on default.



Device description

Node Id:	5
Name:	Bedroom Sealing (5)
Zone:	Bedroom
Brand:	Duewi Z-Wave
Device type:	Routing Multilevel Switch
Product:	Schuko Plug Dimmer
Description:	Schuko Plug-In Dimmer up to 300 WA
To (re-)include:	Tripple Press Button
Documents:	User and Installation Manual
Interview stage:	Interview complete + +++ ++++++
Device state:	✓ Device is operating
Number of packets to be sent:	0
Application version:	1.0
SDK version:	5.01

Switch all configuration

Request current stored values from device

Mode ○ Not in switch all group ○ In switch all off group only ○ In switch all on group only [✓ In switch all on and off groups]
Updated: Thursday, 28 April 17:07

Protection

Request current stored values from device

Local operations [✓ Unprotected] ○ Protection by sequence ○ No operation possible
Updated: Thursday, 28 April 14:36

Figure 7: Demo UI Dialog for Sensors

4. There is more than one device description record matching the information gathered during interview. This is particularly possible if a vendor sells devices with different firmware and functions without properly updating the firmware version information. You may manually choose the correct device description record using the button Select Device Description Record. A dialog box will be opened for manual selection of the product (if available). The manual selection of a device description record is only needed if no record was found on default.

Device Configuration This section will, if device description record is loaded, show device specific configuration values including their possible parameters and a short description of the configuration value. You may change these values according to your needs.

Actions with configurations The most important action in regard to the configuration is to apply the configuration to the device. This is only done when the button Apply configuration for this device is hit. This button is therefore even shown, when the rest of the tab part is hidden.

- Mains Powered Devices: The settings will become effective immediately after hitting the button.
- Battery Powered Devices: The settings will become effective after the next wakeup of the device, as shown in the Device Status tab.
- Battery Powered Controllers (remote controls or wall controllers): The settings will only become effective if the devices are woken up manually. Refer to the controller manual for more information on how to wake up the device. Appendix B may also give further advice.

If there are many similar devices in a network it is desirable to just apply one working configuration to all these devices. This can be done using the function Copy from other Device.

The set of defined configuration values is stored for every device. Therefore its possible to pick a different device and reuse its configuration values for the device to be configured. The Save function of the bottom context menu allows saving the configuration for further use and reuse.

Actions with configurations This function requests the selected device to send its Node Information frame (NIF) to the controller. It can be used instead of triple pressing a button on the device itself that would also instruct the device to send it NIF. The NIF is needed to know device capabilities.

Delete Configuration of this device in section Actions with configuration This function deletes the stored configuration for this device. This function is for debugging purposes only.

Force Interview in section Advanced This functions forces to redo the whole interview. All previous interview data will be deleted. This function is for debugging purposes only.

Show Interview Results in section Advanced This function shows the result of the interview. This function is for debugging purposes only. For information about reasons for incomplete interview please refer to the manual section Device Status.

The bottom context menu function Reset Configuration deletes all configurations stored in Z-Way, but does not affect devices!

The configuration dialog documents the use of the command classes "Association", "Protection", "Configuration" and "Switch All".

4.6.3 Associations

If there was no previous device of the same type installed the interface will show the values as read from the device. If there was already a device of the same kind installed there may exist a stored default configuration for this particular device. Then the setup in the device may differ from the default configuration stored in Z-Way.

- Gray Icon: This Node ID is stored in the device but its not stored in the default configuration of the Z-Way. You can double click this device to store this setting in the Z-Way default configuration of this device type.
- Red Icon: This Node ID is stored locally but not in the association group of the device yet. Apply the settings to transmit the setting into the device. In case of a battery operated device you need to wakeup the device in order to store the configuration.
- Black Icon: This Node ID is stored both in the device and in the local configuration of Z-Way.

Hint: The auto configuration function of Z-Way will place the node ID of the Z-Way controller in all association groups if possible. This allows the activation of scenes from these devices.

All management of Associations is handled by the command class "Association", In case the target device is a multi channel device the command class "Multi Channel Association" is used. Please refer to capter 13 for details how to use these command classes.

5 The Z-Way data model

Z-Way holds all data of the Z-Way network in a data holder structure. The data holder structure is a hierarchical tree of data elements.

Following the object-oriented software paradigm the different commands targeting the network or individual devices are also embedded into the data objects as object methods.

Each data element is handled in a data object that contains the data element and some contextual data.

5.1 The Data object

Each Data element such as devices[nodeID].data.nodeId is an object with the following child elements:

- value: the value itself
- name: the name of the data object
- updateTime: timestamp of the last update of this particular value
- invalidateTime: timestamp when the value was invalidated by issuing a Get command to a device and expecting a Report command from the device
- updated: boolean value indicating if the device is updated or invalid (in time between issuing a Get and receiving a Report)

Every time a command is issues that will have impact on a certain data holder value the time of the request is stored in "invalidateTime" and the "updated" flag is set to "False". This allows to track when a new data value was requested from the network when this new data value was provided by the network.

This is particularly true if Z-Way is sending a SET command. In this case the data value is invalidated with the "SET" commands and gets validated back when the result of the GET command was finally stored in the data model.

To maintain compatibility with Javascript the data object has the following methods implemented

- valueOf(): this allows to obmit .value in JS code, hence write as an example data.level = 255
- updated(): alias to updateTime
- invalidated(): alias to invalidateTime

These aliases are not enumerated if the dataholder is requested (data.level returns value: 255, name: "level", updatedTime: 12345678, invalidatedTime: 12345678).

5.2 The Data and Method Tree

The root of the data tree has two important child objects:

- controller, this is the data object that holds all data and methods (commands, mainly function classes) related to the Z-Way controller as such
- devices array, this is the object array that holds the device specific data and methods (commands, mainly command classes).

Chapter 15 gives a complete overview of the data and method tree.

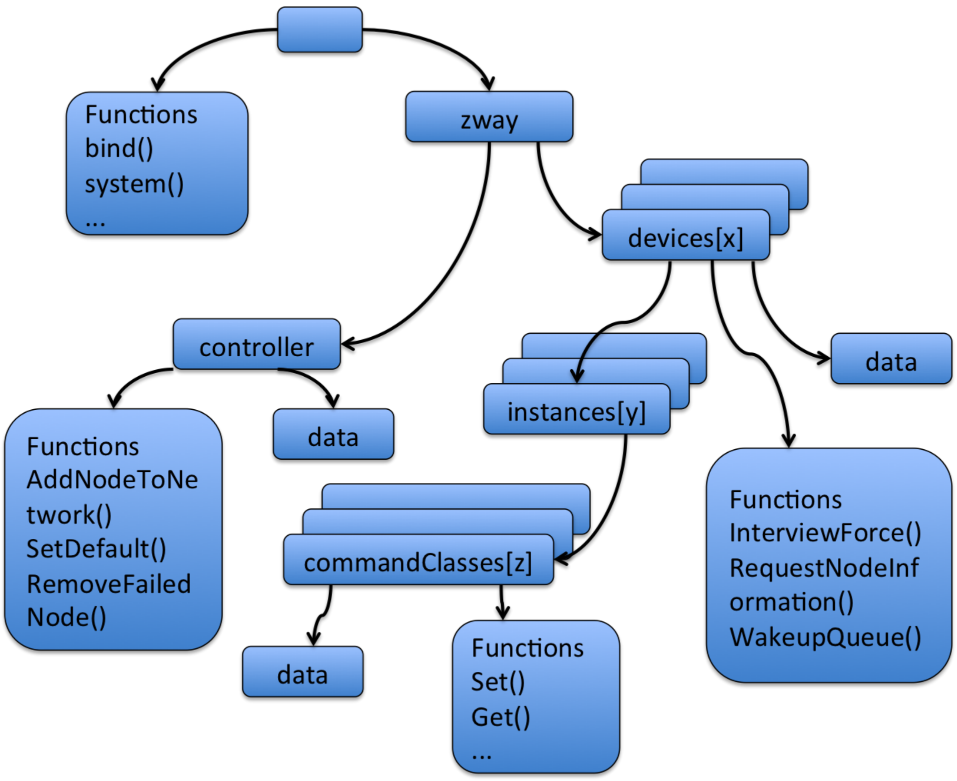
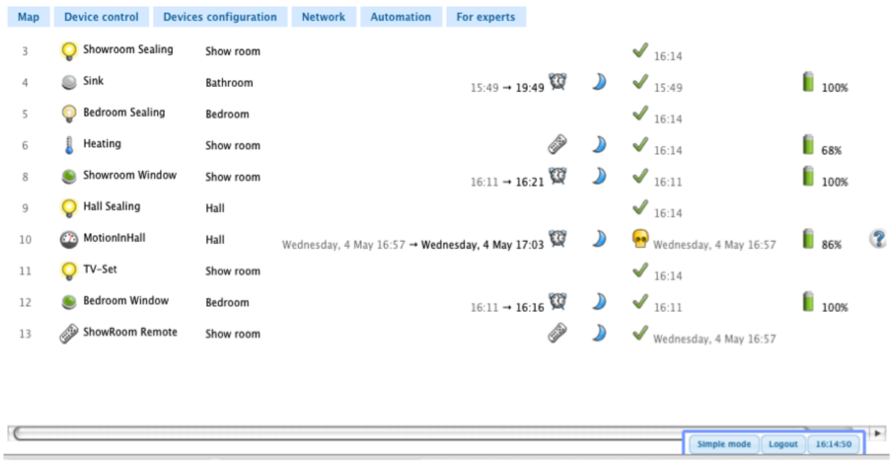


Figure 8: Z-Way Object Tree Structure



3	Showroom Sealing	Show room		✓	16:14		
4	Sink	Bathroom	15:49 → 19:49	✓	15:49	100%	
5	Bedroom Sealing	Bedroom		✓	16:14		
6	Heating	Show room		✓	16:14	68%	
8	Showroom Window	Show room	16:11 → 16:21	✓	16:11	100%	
9	Hall Sealing	Hall		✓	16:14		
10	MotionInHall	Hall	Wednesday, 4 May 16:57 → Wednesday, 4 May 17:03		Wednesday, 4 May 16:57	86%	?
11	TV-Set	Show room		✓	16:14		
12	Bedroom Window	Bedroom	16:11 → 16:16	✓	16:11	100%	
13	ShowRoom Remote	Show room		✓	Wednesday, 4 May 16:57		

Figure 9: Demo UI Dialog for Device Status

5.3 Device Data Vizualization

One example how to use the Data in the object tree is the device status page provided by the demo user interface in tab "Device Control".

This tab gives an overview of the network status and the availability of each device. It shows the time stamp of the last interaction between the controller and the device. For battery powered devices the battery charging status, the time of the last wakeup and the estimated time for the next wakeup is shown. An info icon indicates when the interview of a device was not completed. Clicking on this device opens a window showing the interface status by command class. Please refer to the manual section Interview for more information about the interview process.

The controller information tab shows all controller information. The buttons Show Controller Data shows the internal Z-Way data structure related to the specific controller function of the controller device. The button Show controller device data show the generic device related data of the controller device.

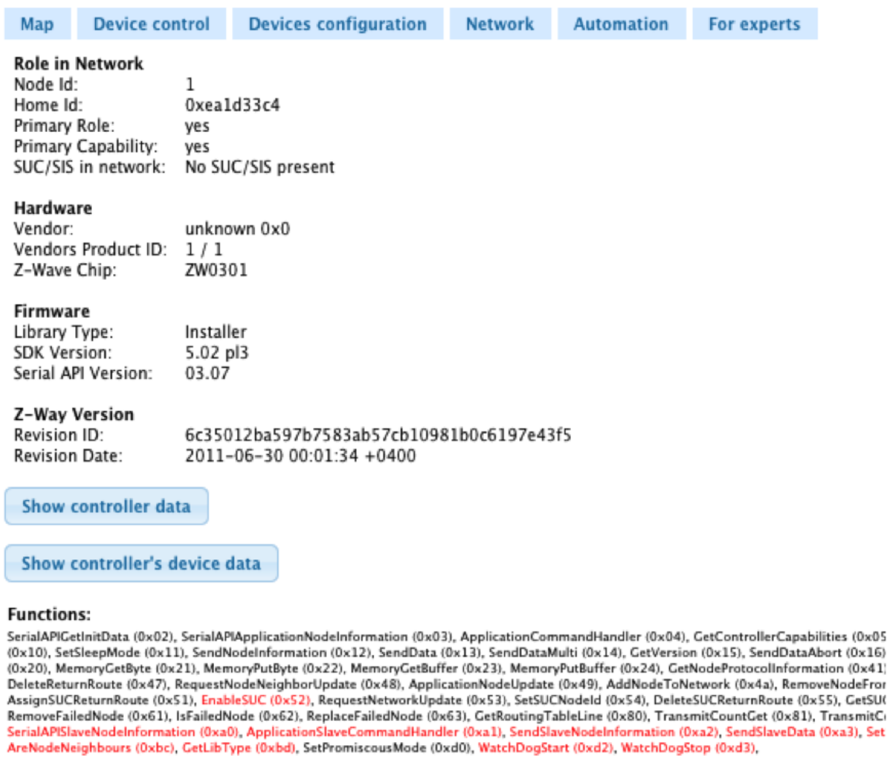
The information given on this page is only relevant for advanced Z-Wave developers and for debugging.

6 Commands to control Z-Way itself

The last set of commands and values are not related to the Z-Wave network or the Z-Wave devices but to Z-Way itself. Chapter 15 lists all the commands and the values.

7 Job Queue Handling

The Job handling system is the core and heart of Z-Way. It is managing the different Function class and command class calls to the Z-Wave network and dispatches incoming messages. Every communication with the Z-Wave transceiver is scheduled into a job and queued that it can transmitted over the serial hardware interface. The API allows to look into the work of the job queue. The demo UI shows the Job queue under Tab "Network" but in expert mode only.



Role in Network

Node Id:	1
Home Id:	0xea1d33c4
Primary Role:	yes
Primary Capability:	yes
SUC/SIS in network:	No SUC/SIS present

Hardware

Vendor:	unknown 0x0
Vendors Product ID:	1 / 1
Z-Wave Chip:	ZW0301

Firmware

Library Type:	Installer
SDK Version:	5.02 pl3
Serial API Version:	03.07

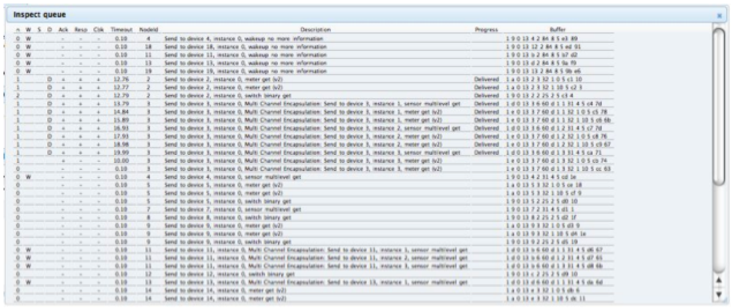
Z-Way Version

Revision ID:	6c35012ba597b7583ab57cb10981b0c6197e43f5
Revision Date:	2011-06-30 00:01:34 +0400

Functions:

SerialAPIGetInitData (0x02), SerialAPIApplicationNodeInformation (0x03), ApplicationCommandHandler (0x04), GetControllerCapabilities (0x05
(0x10), SetSleepMode (0x11), SendNodeInformation (0x12), SendData (0x13), SendDataMulti (0x14), GetVersion (0x15), SendDataAbort (0x16)
(0x20), MemoryGetByte (0x21), MemoryPutByte (0x22), MemoryGetBuffer (0x23), MemoryPutBuffer (0x24), GetNodeProtocolInformation (0x41)
DeleteReturnRoute (0x47), RequestNodeNeighborUpdate (0x48), ApplicationNodeUpdate (0x49), AddNodeToNetwork (0x4a), RemoveNodeFror
AssignSUCReturnRoute (0x51), EnableSUC (0x52), RequestNetworkUpdate (0x53), SetSUCNodeId (0x54), DeleteSUCReturnRoute (0x55), GetSU(
RemoveFailedNode (0x61), IsFailedNode (0x62), ReplaceFailedNode (0x63), GetRoutingTableLine (0x80), TransmitCountGet (0x81), TransmitC
SerialAPISlaveNodeInformation (0xa0), ApplicationSlaveCommandHandler (0xa1), SendSlaveNodeInformation (0xa2), SendSlaveData (0xa3), Set
AreNodeNeighbours (0xbc), GetLibType (0xbd), SetPromiscousMode (0xd0), WatchDogStart (0xd2), WatchDogStop (0xd3),

Figure 10: Demo UI Dialog for Controller information



Inspect queue

n	W	S	D	Ack	Resp	Cbk	Timeout	NodeId	Description	Progress	Buffer
0	W			-	-	-	0.10	4	Send to device 4, instance 0, wakeup no more information		1 9 0 13 4 2 84 8 5 e3 89
0	W			-	-	-	0.10	18	Send to device 18, instance 0, wakeup no more information		1 9 0 13 12 2 84 8 5 ed 91
0	W			-	-	-	0.10	11	Send to device 11, instance 0, wakeup no more information		1 9 0 13 b 2 84 8 5 b7 d2
0	W			-	-	-	0.10	13	Send to device 13, instance 0, wakeup no more information		1 9 0 13 d 2 84 8 5 9a f9
0	W			-	-	-	0.10	19	Send to device 19, instance 0, wakeup no more information		1 9 0 13 13 2 84 8 5 9b e6
1			D	+	+	+	12.76	2	Send to device 2, instance 0, meter get (v2)	Delivered	1 a 0 13 2 3 32 1 0 5 c1 10
1			D	+	+	+	12.77	2	Send to device 2, instance 0, meter get (v2)	Delivered	1 a 0 13 2 3 32 1 10 5 c2 3
2			D	+	+	+	12.79	2	Send to device 2, instance 0, switch binary get	Delivered	1 9 0 13 2 2 25 2 5 c3 4
1			D	+	+	+	13.79	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 1, sensor multilevel get	Delivered	1 d 0 13 3 6 60 d 1 1 31 4 5 c4 7d
1			D	+	+	+	14.84	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 1, meter get (v2)	Delivered	1 e 0 13 3 7 60 d 1 1 32 1 0 5 c5 78
1			D	+	+	+	15.89	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 1, meter get (v2)	Delivered	1 e 0 13 3 7 60 d 1 1 32 1 10 5 c6 6b
1			D	+	+	+	16.93	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 2, sensor multilevel get	Delivered	1 d 0 13 3 6 60 d 1 2 31 4 5 c7 7d
1			D	+	+	+	17.93	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 2, meter get (v2)	Delivered	1 e 0 13 3 7 60 d 1 2 32 1 0 5 c8 76
1			D	+	+	+	18.98	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 2, meter get (v2)	Delivered	1 e 0 13 3 7 60 d 1 2 32 1 10 5 c9 67
1			D	+	+	+	19.99	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 3, sensor multilevel get	Delivered	1 d 0 13 3 6 60 d 1 3 31 4 5 ca 71
1				+	-	-	10.00	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 3, meter get (v2)		1 e 0 13 3 7 60 d 1 3 32 1 0 5 cb 74
0				-	-	-	0.10	3	Send to device 3, instance 0, Multi Channel Encapsulation: Send to device 3, instance 3, meter get (v2)		1 e 0 13 3 7 60 d 1 3 32 1 10 5 cc 63
0	W			-	-	-	0.10	4	Send to device 4, instance 0, sensor multilevel get		1 9 0 13 4 2 31 4 5 cd 1e
0				-	-	-	0.10	5	Send to device 5, instance 0, meter get (v2)		1 a 0 13 5 3 32 1 0 5 ce 18
0				-	-	-	0.10	5	Send to device 5, instance 0, meter get (v2)		1 a 0 13 5 3 32 1 10 5 cf 9
0				-	-	-	0.10	5	Send to device 5, instance 0, switch binary get		1 9 0 13 5 2 25 2 5 d0 10
0				-	-	-	0.10	7	Send to device 7, instance 0, sensor multilevel get		1 9 0 13 7 2 31 4 5 d1 1
0				-	-	-	0.10	8	Send to device 8, instance 0, switch binary get		1 9 0 13 8 2 25 2 5 d2 1f
0				-	-	-	0.10	9	Send to device 9, instance 0, meter get (v2)		1 a 0 13 9 3 32 1 0 5 d3 9
0				-	-	-	0.10	9	Send to device 9, instance 0, meter get (v2)		1 a 0 13 9 3 32 1 10 5 d4 1e
0				-	-	-	0.10	9	Send to device 9, instance 0, switch binary get		1 9 0 13 9 2 25 2 5 d5 19
0	W			-	-	-	0.10	11	Send to device 11, instance 0, Multi Channel Encapsulation: Send to device 11, instance 1, sensor multilevel get		1 d 0 13 b 6 60 d 1 1 31 4 5 d6 67
0	W			-	-	-	0.10	11	Send to device 11, instance 0, Multi Channel Encapsulation: Send to device 11, instance 2, sensor multilevel get		1 d 0 13 b 6 60 d 1 2 31 4 5 d7 65
0	W			-	-	-	0.10	11	Send to device 11, instance 0, Multi Channel Encapsulation: Send to device 11, instance 3, sensor multilevel get		1 d 0 13 b 6 60 d 1 3 31 4 5 d8 6b
0				-	-	-	0.10	12	Send to device 12, instance 0, switch binary get		1 9 0 13 c 2 25 2 5 d9 10
0	W			-	-	-	0.10	13	Send to device 13, instance 0, Multi Channel Encapsulation: Send to device 13, instance 1, sensor multilevel get		1 d 0 13 d 6 60 d 1 1 31 4 5 da 6d
0				-	-	-	0.10	14	Send to device 14, instance 0, meter get (v2)		1 a 0 13 e 3 32 1 0 5 db 6
0				-	-	-	0.10	14	Send to device 14, instance 0, meter get (v2)		1 a 0 13 e 3 32 1 10 5 dc 11

Figure 11: Job Queue Vizualization in Demo UI

n	This column shows the number of sending attempts for a specific job. Z-??Way tries three times to dispatch a job to the transceiver.
W,S,D:	This shows the status of the job. If no indicator is shown the job is in active state. This means that the controller just tries to execute the job. 'W' states indicated that the controller believes that the target device of this job is in deep sleep state. Jobs in 'W' state will remain in the queue to the moment when the target devices announces its wakeup state by sending a wakeup notification to the controller. Jobs in 'S' state remain in the waiting queue to the moment the security token for this secured information exchanged was validated. 'D' marks a job as done. The job will remain in the queue for information purposes until a job garbage collection removed it from the queue.
ACK:	shows if the Z-??Wave transceiver has issued an ACK message to confirm that the message was successfully received by the transceiver. This ACK however does not confirm that the message was delivered successfully. A successful delivery of a message will result in a D state of this particular job. If the ACK field is blank, then no ACK is expected. A . indicates that the controller expects an ACK but the ACK was not received yet. A + indicates that an ACK was expected and was received.
RESP	shows if a certain command was confirmed with a valid response. Commands are either answered by a response or a callback. If the RESP field is blank, then no Response is expected. A . indicates that the controller expects a Response but the Response was not received yet. A + indicates that a Response was expected and was received.
Cbk	If the Cbk field is blank, then no callback is expected. A . indicates that the controller expects a Callback but the Callback was not received yet. A + indicates that a Callback was expected and was received.
Timeout	Ishows the time left until the job is de queued
Node Id	shows the id of the target node. Communication concerning the network like inclusion of new nodes will have the controller node id as target node ID. For command classes command the node ID of the destination Node is shown. For commands directed to control the network layer of the protocol, the node id is zero.
Description	shows a verbal description of the job
Progress	shows a success or error message depending on the delivery status of the message. Since Z-??Way tries three times to deliver a job up to 3 failure messages may appear. Buffer: ... shows the hex values of the command sent within this job

Table 1: Parameters of the Job Queue Vizualization

The table shows the active jobs with their respective status and additional information.

Table 1 summarizes the different values displayed on the Job Queue visualization. While these infos are certainly not relevant for end users of the system it is a great debug tool.

8 JSON-API

JSON API allows to execute commands on server side using HTTP POST requests (currently GET requests are also allowed, but this might be deprecated in future). The command to execute is taken from the URL.

All functions are executed in form

http://IPOFYOURRASPBERRYPI:8083/<URL>

8.1 /ZWaveAPI/Run/<command>

This executes zway.<command> in JavaScript engine of the server. As an example to switch ON a device no 2 using the command class BASIC (0x20) its possible to write:

```
/ZWaveAPI/Run/devices[2].instances[0].commandClasses[0x20].Set(255)
```

or

```
/ZWaveAPI/Run/devices[2].instances[0].Basic.Set(255)
```

The Z-Way demo GUI has a JS command runCmd(<command>) to simplify such operations. This function is accessable in the Javascript console of your web browser (in Chrome you find the JavaScript console unter View->Debug->JS Console). Using this feature the command in JS console would looke like

```
runCmd('devices[2].instances[0].Basic.Set(255)')
```

The usual way to access a command class is using the format 'devices[nodeId].instances[instanceId]. commandClasses[commandclassId]'. There are ways to simplify the syntax:

- 'devices[nodeId].instances[instanceId].commandClasses.Basic' is equivalent to 'devices[nodeId]. instances[instanceId].commandClasses[0x20]'
- The word 'commandClass' can be obmitted if the command class name is used: 'devices[nodeId]. instances[instanceId].Basic' is equivalent to 'devices[nodeId]. instances[instanceId]. commandClasses.Basic'
- the instances[0] can be obmitted: 'devices[nodeId].instances[instanceId].Basic' then turns into 'devices[nodeId]. Basic'

Each Instance object has a device property that refers to the parent device it belongs to. Each Command Class Object has a device and an instance property that refers to the instance and the device this command class belongs to.

Data holder object have properties value, updateTime, invalidateTime, name, but for compatibility with JS and previous versions we have valueOf() method (allows to omit .value in JS code, hence write "data.level == 255"), updated (alias to updateTime), invalidated (alias toinvalidateTime). These aliases are not enumerated if the dataholder is requested (data.level returns value: 255, name: "level", updatedTime: 12345678, invalidatedTime: 12345678).

8.2 /JS/Run/<command>

Executes <command> in JavaScript engine of the server and returns JSON object returned by JavaScript function.

For example to execute a request for binary sensor value every 5 minutes use:

> /JS/Run/setInterval(function() { zway.devices[2].instances[0].commandClasses[0x30].Get(); }, 300*1000);

Note: /ZWaveAPI/Run/<command> is an alias to /JS/Run/zway.<command>

The Z-Way demo GUI has a JS command runJS(<command>) to simplify such operations. This function is accessable in the Javascript console of your web browser (in Chrome you find the JavaScript console unter View->Debug->JS Console). Using this feature the command in JS console would looke like

> runJS('setInterval(function() {zway.devices[2].instances[0].commandClasses[0x30].Get(); }, 300*1000);')

8.3 /ZWaveAPI/InspectQueue

This function is used to vizualize the Z-Way job queue. This is for debugging only but very useful to understand the current state of Z-Way engine.

8.4 /ZWaveAPI/Data/<timestamp>

Returns an associative array of changes in Z-Way data tree since <timestamp>. The array consists of (<path>: <JSON object>) object pairs. The client is supposed to assign the new <JSON object> to the subtree with the ¡path¿ discarding previous content of that subtree. Zero (0) can be used instead of <timestamp> to obtain the full Z-Way data tree.

The tree have same structure as the backend tree (Figure 8) with one additional root element "updateTime" which contains the time of latest update. This "updateTime" value should be used in the next request for changes. All timestamps (including updateTime) corresponds to server local time.

Each node in the tree contains the following elements:

- value the value itself
- updateTime timestamp of the last update of this particular value
- invalidateTime timestamp when the value was invalidated by issuing a Get command to a device and expecting a Report command from the device

The object looks like:

Listing 1: JSON Data Structure

```
{
"[path_from_the_root]": [updated subtree],
"[path_from_the_root]": [updated subtree],
...
updateTime: [current timestamp]
}
```

Examples for Commands to update the data tree look like:

Get all data: /ZWaveAPI/Data/0

Get updates since 134500000 (Unix timestamp): /ZWaveAPI/Data/134500000

Note that during data updates some values are update by big subtrees. For example, in Meter Command Class value of a scale is always updated as a scale subtree by [scale].val object (containing scale and type descriptions).

8.5 Handling of updates coming from Z-Way

A good design of a UI is linking UI objects (label, textbox, slider, ...) to a certain path in the tree object. Any update of a subtree linked to UI will then update the UI too. This is called bindings.

For web applications Z-Way Web UI contains a library called jQuery.triggerPath (extention of jQuery written by Z-Wave.Me), that allows to make such links between objects in the tree and HTML DOM objects. Use

var tree;

jQuery.triggerPath.init(tree);

during web application initialization to attach the library to a tree object. Then run

jQuery([objects selector]).bindPath([path with regexp], [updater function], [additional arguments]);

to make binding between path changes and updater function. updater function would be called upon changes in the desired object with this pointing to the DOM object itself, first argument pointing to the updated object in the tree, second argument is the exact path of this object (fulfilling the regexp) and all other arguments copies additional arguments. RegExp allows only few control characters: * is a wildcard, (1—2—3) - is 1 or 2 or 3. For example:

Listing 2: Subscription of Data Values

```
$('span.light_level').bindPath('(devices[*].instances[0].commandClasses[38].
data.level|devices.*.instances.*.commandClasses[39].data.level)',
function (obj, path, arg1, arg2) \{
        $(this).html(path + ': ' + obj.value);
        \}, "this will be passed to update parser", "this too");
```

And finally here is an example of function handling updates (it is supposed to be called each 1-10 seconds):

Listing 3: Data Update

```

// Call this function periodically or manually to get updates.
// Parameter sync allows to make request synchronous or asynchronous

function getDataUpdate(sync) {
        $.postJSON('/ZWaveAPI/Data/' + tree.updateTime, handlerDataUpdate, sync);
        }

```

```
// Wrapper that makes AJAX request
$.postJSON = function(url, data, callback, sync) {
        // shift arguments if data argument was omited
        if (jQuery.isFunction(data)) {
                sync = sync || callback;
                callback = data;
                data = {};
        };
        $.ajax({
                type: 'POST',
                url: url,
                data: data,
                dataType: 'json',
                success: callback,
                error: callback,
                async: (sync!=true)
                });
        };

// Handle updates
function handlerDataUpdate(data) {
try {
        $.each(data, function (path, obj) {
                var pobj = tree;
                var pe_arr = path.split('.');
                for (var pe in pe_arr.slice(0, -1)) {
                        pobj = pobj[pe_arr[pe]];
                        };
                pobj[pe_arr.slice(-1)] = obj;

// Restrict UI updates only to some paths. This is optional to make parsing faster.
// Remove this "if" to parse all updates
        if (
        (new RegExp('^devices\\..*\\.instances\\..*\\.commandClasses\\.
␣␣␣␣␣␣␣␣(37|38|48|49|50|67|98|128)\\.data.*$')).test(path) || (new RegExp(
        '^areas\\.data\\..*\\..*$')).test(path))
        $.triggerPath.update(path); // this updates objects bound to paths
        });
} catch(err) {
        console.log("Error␣in␣handlerDataUpdate:␣" + err.stack);
};
};
```

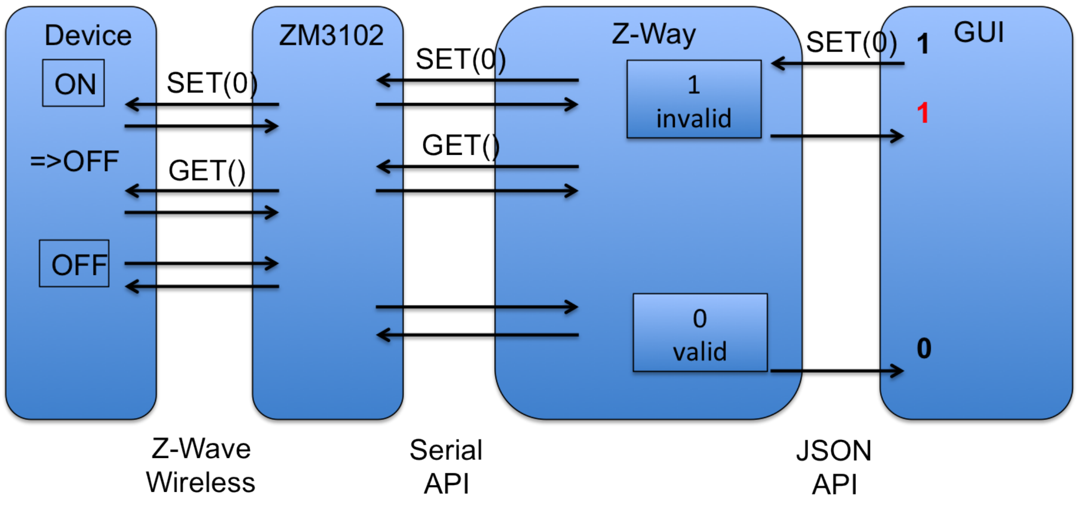


Figure 12: Z-Way Timings

9 Executing a command from the GUI to the device and back

To transport data between the real wireless device and the GUI multiple communication instances are involved. The complexity of this communication chain shall be explained in the following example:

Assuming the GUI shows the status of a remote switch and allows to change the switching state of this device. When the users hits the switching button he expects to see the result of his action as a changing status of the device in the GUI. The first step is to hand over the command (SET) from the GUI to Z-Way using the JSON interface. Z-Way receives the command and will confirm the reception to the GUI. Z-Way recognizes that the execution of the switching common will likely result in a change of the status variable However Z-Way will not immediately change the status variable but invalidate the actual value. this is the correct action because at the moment when the command was received the status is the remote device has not been changed. Since it will the status of the switch is unknown. If the GUI polls the value it will still see the old value but marked as invalid. Z-Way will not hand over the switching command to the Z-Wave transceiver chip. Since it is possible that there are other command waiting for execution (sending) by the Z-Wave transceiver chip the job queue is queuing them and will handle certain priorities if needed. Z-Way has recognized that the command will likely change the status of the remote device and is therefore adding another command to call the actual status after the switching command was issued. The transceiver is confirming the reception of the command and this confirmation is noted in the job queue. This confirmation however only means that the transceiver has accepted the command and does neither indicate that the remote device has receives it nor even confirming that the remote device has executed accordingly. The transceiver will now try to send the commend wirelessly to the remote device. A successful confirmation of the reception from the remote device is the only valid indicator that the remote device has received the command (again, not that it was executed!). The second command (GET) is now transmitted the very same way and confirmed by the remote device. This device will now sent a

REPORT command back to Z-Way reporting the new status of the switching device. Now the transceiver of Z-Way has to confirm the reception. The transceiver will then send the new value to the Z-Way engine by issuing a commas via the serial interface. Z-Way receives the report and will update the switching state and validate the value. From now on the GUI will receive a new state when polling.

10 Automation Engine

The automation engine performs different actions based on events. The actions are either signal commands or scripts that can add additional logic and conditions. Events, either generated from the Z-Wave network or from an outside sources such as the Internet or even from a user interaction is causing certain actions, either within the Z-Wave network (e.g. switching a light) or outside Z-Wave (e.g. sending a email). In Z-Way all automation is organized in so called modules. The subsequent manual will explain how these modules work and how to create own modules.

Attention: There is no Graphical User Interface for the Automation engine at this moment in time. You will need a text editor such as joe, textwrangler or vi to edit certain files.

10.1 How to get to the automation engine

The starting point for automation in Z-Way is called config.xml and is located in the main folder of Z-Way. The statement for the automation engine looks like

$$< automation - dir > pathToAutomationCodeBase < /automation - dir >$$

Assuming the automation is like on default in the subdirectory /automation the statement should look like

$$< automation - dir > automation < /automation - dir >$$

The automation folder consists of several files and subdirectories. The most important file of the automation is called config.json. On default there is no config.json but you may create an initial configuration file from the template config-sample.json

Lets have a look into this file. It consists of one JSON object description, that defines certain basic configuration settings:

basePath: This path variable points to root of the automation engine. In most cases this is './'

controller These are some definition of setups of the controller itself. Within the controller definition there are three further sections:

- modulePath: the folder where all the automation modules are stored
- modules: a list of all the modules that shall be known to the controller
- instances: the active instances of the modules

The listing below shows a sample configuration file

Listing 4: Sample JSON Config File

```
{
  "basePath": "./,
  "controller": {
    "modulesPath": "./modules",
     "modules": [
       "EventLog",
       "ZWaveGate",
       "AutoOff"
       ],
      "instances": {
        "EventLog" : {
          "module": "EventLog" ,
          "config": {}
          },
        "ZWave" : {
          "module": "ZWaveGate" ,
          "config": {}
            }
        }
    }
}
```

The module variable lists all the modules separates by comma that shall be known to the controller. The sample config only shows the modules 'EventLog', 'ZWaveGate' and 'AutoOff'. Whenever a new module is created it needs to be added here in order to be registered. Please note that listing a module here only make the module available to the automation engine but will not instantiate the functionality.

The next part finally instantiates the module within the structure 'instances' The sample code shows the two instance, one of the module 'Eventlog' and one of the Module 'ZWaveGate'.

From the sample code we can learn

- that every instance has a unique name and the module t obe instantiated is defines in the varable module
- the very same module can be instantiated multiple times but with different names
- not every module needs to be instantiated.

Every instance consist of the name of the instance, the reference to the module (that was declared in the upper part of the config file and a config structure. The config structure allows to send certain configuration values to the module.

Example: Assuming there is a module that turns on a light at a certain time, accepting the node ID and the time for action, then there could be multiple instances with different node IDs and times as config parameter.

Listing 5: Another Sample Config File - Instance Part

```
instances {
 TurnOn Floor  {
         module     TurnOn  ,
              config{   node  : 2,
                              time  :   22:00   }},
 TurnOn Kitchen  {
         module  :   TurnOn  ,
              config{   node  : 5,
                              time  :   18:00   }},
}
```

The modules will be instantiated one after each other in the order of its entries in the instances section.

Warning! The current Z-Way Home Automation system requires 'ventLog' and 'ZWave' modules to be always instantiated. 'EventLog should always be instantiated as first module prior to any other module. 'The module 'EventLog' creates the event logging subsystem and the module "ZWave" enables the Z-Wave network bindings.

10.2 The Event Bus

All communication from and to the automation modules is handled by events. An event is a structure containing certain information that is exchanged using a central distribution place, **the event bus**. This means that all modules can send events to the event bus and can listen to event in order to execute commands on them. All modules can 'see' all events but need to filter out heir events of relevance. The core objects of the automation are written in JS and they are available as source code in the sub folder classes:

- AutomationController.js: This is the main engine of the automation function
- AutomationModule.js: the basic object for the module
- VirtualDevice.js: The implementation of the virtual device that is the abstraction of all real device but the way to create new devices not related to certain physical Z-Wave devices.

The file main.js is the startup file for the automation system and it is loading the three classes just mentioned. The subfolder /lib contains the key JS script for the Event handling: eventemitter.js.

10.2.1 Emitting events

Eventemitter emits events into the central event bus. The event emitter can be called from all modules and scripts of the automation system. The syntax is:

$$controller.emit(eventName, args1, arg2, ...argn)$$

The event name 'eventName' has to be noted in the form of XXX.YYY where XXX is the name of the event source (e.g. the name of the module issuing the event or the name of the module using the event) and YYY is the name of the event itself. To allow a scalable system it makes sense to name the events by the name of the module that is supposed to receive the and to manage events. This simplifies the filtering of these events by the receiver module(s).

Certain event names are forbidden for general use because they are already used in the existing modules. One example are events with the name cron.XXXX that are used by the cron module handling all timer related events.

Every event can have a list of arguments developers can decide on. For the events used by preloaded modules (first and foremost the cron module) this argument structure is predefined. For all other modules the developer is free to decide on structure and content. It is also possible to have list fields and or any other structure as argument for the event

One example of an issued event can be

$$emit(mymodule.testevent, Test, [event1, event2])$$

10.2.2 Catching events

The controller object, part of every module, offers a function called 'on()' to catch events. The 'on(name, function())' function subscribes to events of a certain name type. If not all events of a certain name tag shall be processed a further filtering needs to be implemented processing the further arguments of the event. The function argument contains a reference to the implementation using the event to perform certain actions. The argument list of the event is handed over to this function in its order but need to be declared in the function call statement.

```
this.controller.on(mymodule.testevent, function (name,eventarray) )
```

10.3 Module-Syntax

Each module is located in a sub directory of the module-subfolder defined in the config.json file. The name of the sub folder equals to the module name (not the instance of the module name!) and has at least two files:

10.3.1 Module.json

This file contains the module meta-definition used by the AutomationController. It must be a valid JSON object with the following fields (all of them are required):

- autoload Boolean, defines will this module automatically instantiated during Home Automation startup.
- singleton Boolean, defines this module can be instantiated more than one time or not.
- defaults Object, default module instance settings. This object will be patched with the particular config object from the controller's configuration and resulting object will be passed to the initializer.

All configuration fields are required. Types of the object must equals definition in every case. For instance, if module doesn't export any metric corresponding key value should be and empty object .

10.3.2 index.js

This script defines an automation module class which is descendant of AutomationModule base class. During initialization the module script must define the variable '_module' containing the particular module class.

Example of a minimal automation module:

Listing 6: Minimal Module

```

function SampleModule (id, controller) {
    SampleModule.super_.call.init(this, id, controller);

    this.greeting = "Hello, World!";
}

inherits(SampleModule, AutomationModule);
_module = SampleModule;
```

```

SampleModule.prototype.init = function () {
    this.sayHello();
}

SampleModule.prototype.sayHello = function () {
    debugPrint(this.greeting);
}
```

The first part of the code illustrates how to define a class function named SampleModule that calls the superclass' constructor. Its highly recommended not to do further instantiations in the constructur. Initializations should be implemented within the 'init' function.

The second part of the code is almost immutable for any module. It calls prototypal inheritance support routine and it fills in _module variable.

The third part of the sample code defines module's init() method which is an instance initializer. This initializer must call the superclass's initializer prior to all other tasks. In the initializer module can setup it's private environment, subscribe to the events and do any other stuff. Sometimes, whole module code can be placed withing the initializer without creation of any other class's methods. As the reference of such approach you can examine AutoOff module source code.

After the init function a module may contain other functions. The 'sayHello' function of the Sample Module shows this as example.

10.4 Available Core Modules

The automation engine already contains certain modules essential for the work of the whole system. Do now exclude these modules from the config.json and alter them only if you know exactly what you do.

10.4.1 Cron, the timer module

All time driven actions need a timer. The Z-Way automation engine implement a cron-type timer system as a module as well. The basic function of the cron module is

- It accepts registration of events that are triggered periodically
- It allows to de-register such events.evice that is the abstraction of all real device but the way to create new devices not related to certain physical Z-Wave devices.

The registration and deregistration of events is also handled using the event mechanism. The cron module is listening for events with the tags 'cron.addTask' and 'cron.removeTask'. The first argument of these events are the name of the event fired by the cron module. The second argument of the 'addTask' event is an array desricing the times when this event shall be issued. It has the format:

- Minute [start,stop, step] or 0-59 or null
- Hour [start,stop, step] or 0-23 or null
- weekDay [start,stop, step] or 0-6 or null
- dayOfMonth [start,stop, step] or 1-31 or null

- Month [start,stop, step] or 1-12 or null

The argument for the different time parameters has one of three formats

- null: the event will be fired on every minute or hour etc.
- single value: the event will be fired when the value reaches the given value
- array [start, stop, step]: The event will be fired between start and stop in steps.

The object

$$\{minute : null, hour : null, weekDay : null, day : null, month : null\}$$

will fire every minute within every hour within every weekday on every day of the month every month. Another example of an event emitted towards the cron module for registering an timer event can be found in the Battery Polling Module:

Listing 7: Registering a Battery Polling Command

```
    this.controller.emit("cron.addTask", "batteryPolling.poll", {
        minute: 0,
        hour: 0,
        weekDay: this.config.launchWeekDay,
        day: null,
        month: null
    });
```

This call will cause the cron module to emit an event at night night (00:00) on a day that is defines in the configuration variable this.config.launchWeekDay, e.g. 0 = Sunday.

The cron.removeTask only needs the name of the registered event to deregister.

10.4.2 Z-Wave

All Z-Wave related functions of Z-Way are encapsulated in the zway object and can be accessed directly from all automation modules in the same wayas described in the JSON interface. However it is recommended to access z-Wave related functions using the virtual device associated wit the Z-Wave devices.

The next chapter explains the concept and use of virtual devices.

10.5 Virtual Devices

As long as the automation functions only deal with commands and data of dedicated physical device of one wireless technology like Z-Wave its possible to only use these physical devices accessible using the 'zway' data object.

At the moment when other data like information from the internet or from other wireless technology stacks shall used an abstraction layer is needed that unifies the access and usage of data and command. Z-Ways introduces the concept of virtual devices 'vDev'

The virtual device is a data object than can (or not) relate to real data of physical devices. It can also aggregate data of different physical devices or create own data and offer own methods. Virtual devices are created by the automation modules by inheriting them from the base class 'VirtualDevice'. This base class is quite simple. It has a list of class variables plus two public function to be overwritten by the implementation of a virtual device.

Listing 8: Base Class of Virtual Devices

```
VirtualDevice = function (id, controller) {
    this.id = id;
    this.controller = controller;
    this.deviceType = null;
    this.metrics = {};
}

VirtualDevice.prototype.setMetricValue = function (name, value) {
    this.metrics[name] = value;
    this.controller.emit("metricUpdate."+this.id, name, value);
}

VirtualDevice.prototype.performCommand = function (command) {
    console.log("--- VirtaulDevice.performCommand", command);
    return false;
}
```

10.5.1 Metrics

To create a data element in a virtual device the object method 'setMetricValue(name, value)' of the instance of the virtual device must be called. This method has two arguments:

- Name: This String is the name of the variable the data value can be referred.
- Value: This variable of type 'mixed' contains the data element.

Whenever a metric (say a value of the device) is changing, either by changing the value from code of the module of by changing from a binding toa physical device, the virtual device is emitting a 'metricUpdate.vDevId' message to the event bus with the arguments 'name' and 'newValue'. Within the event name 'vDevId' is the name of the virtual device.

10.5.2 Commands

Any virtual device exposes the method 'performCommand()' with one argument: the command name as String. This function can be overwritten by any virtual device implementation and filled with the implementation of real commands related to the desired function of the virtual device. The return value shall be true or false depending on the success of the command issued.

10.5.3 Z-Wave Bindings

The Z-Wave subsystem creates a virtual device for every physical device. The implementation, variables and functions of the virtual devices created depend on the functions and values of the physical device. All Virtual Devices created by physical Z_Wave devices are managed by the module 'ZWaveGate'. This module contain an array 'instanceDevices' what holds all the Z-Wave related virtual devices. The virtual devices are either generic Z-Wave devices as defined in the file '/modules/ZWaveGate/classes/ZWaveDevice.js' or function specific implementation of virtual devices such as e.g. 'ZWaveSwitchMultiLevel.js' in the same sub folder that inherits the functions of 'ZWaveDevice' inheriting the interfaces of VirtualDevice.

Every virtual device creates from a physical Z-Wav device has a reference to this pysical device as instance variables:

- zDeviceId
- zInstanceId
- zCommandClassId
- zSubTreeId

The ZWave module will also emit events on every update of the z-way data elements (zway.dataUpdate) and on every change of the zway data structure (zway.structureUpdate).

10.6 Additional Javascript Commands useful for automation

The z-way automation engine is based on Javascript as specified in the ECMA specification document

http://www.ecma-international.org/publications/standards/Ecma-262.htm

This implementation does not offer any automated module loading functions. Every Javascript code runs in the same global name space. Therefore its important to maintain certain naming conventions and keep the namespace as clean as possible.

It must also be understood that the implementation running on the backend of zway does NOT offer all the nice library and communication functions of a web browsers Java script implementation. In particular there are not direct function to access network resources and not function to access the local filesystem.

The following functions implemented in the zway automation system need to be used instead:

10.6.1 loadJSON(filename)

This function reads a file from the filesystem and loads it into the memory. The file must contain a valid JSON object. The only argument is the name of the file including full pathname of the local filesystem. The functions returns the full JSON object or null in case of error.

10.6.2 executeFile(filename)

Loads and executes a particular javascript file from the local filesystem. The script is executed withig the global namespace.

Remark: If an error occurred during the execution it won't stop callee from further execution, but erroneous script will not be executed completely. It will stop on the first error.

10.6.3 system(command)

The command system() allows to execute any shell level command available on the operating system. It will return the shell output of the command. One common example of this function would be the call of a 'wget' function to access data from internet services. On default the execution of system commands is forbidden. Each command executed need to be permitted by putting one line with the starting commands in the file automation/.syscommands or in an different automation folder as specified in config.xml.

10.6.4 Debugging

Please us the functions 'debugPrint()' or 'console.log()' for debugging purposes.

Further function calls and descriptions can be found in the Z-Way Data Reference , particularly in the root portion of it.

11 File System

All Z-Way files of RaZberry are located in the folder /data and its subfolders

11.1 config.xml

This file contains the global settings to get the code running

- device: The serial device of the Transceiver chip. For RaZberry this is /dev/AMA0
- config-dir: Optioh to change folder of config files
- translations-dir: Optioh to change folder of translations files
- zddx-dir: Option to change folder of zddx files
- port: port of the web server
- http-root-dir: root file system of web server
- automation-file: name of automation main faile
- shell-script-on-save-xml: points to shell script for backup

11.2 Subfolder /ZDDX

This folder contains the Device Description Records (DDR). These DDRs are human readable XML files providing device information not accessible from the device it self. Z-Way determines the correct Device Type identified by the Vendor ID and vendor specific Product IDs. The XML files contain among others verbal descriptions of the association groups and configuration parameters.

11.3 Subfolder config

This subfolder contains installation specific settings:

- File Configuration.xml: The default configuration values of different device types. This is autogenerated.
- File Defaults.xml: Some basic definitions of Z-Way. This may need manual changes.
- File Rules.xml: user defined device names, maps, etc. This is autogenerated.
- Folder Maps: The pictures of the floor plan
- Folder zddx: the device data of included devices plus the whole device data tree .

11.3.1 Options of File Defaults.xml

- AutoConfig: Flag if Z-Way shall interview the device right after inclusion (default = 1)
- DeepInterview: Flag that Interview is only completed after all values are received back. This includes Asking the device for all initial values of sensor or status data (default = 1)
- SaveDataAfterInterviewSteps: Flag whether or not all device data shall be saved after each interview step (default = 1)
- TryToBecomeSIS: Shall Z-Way try to become Networks SIS if transceiver hardware allows to (default = 1)
- Controller: Description how Z-Way shall behave as device in the network. This entry has the following subentries
  - NodeInformationFrame: The command classes Z-Way is announcing as "supported" in the network
  - SecureNodeInformationFrame: Command Classes available in secure environment
  - InstanceNodeInformationFrame: Command Classes in Instances if Multi Channel is emulated
  - Version id =iD: Versions to be reported in Command Class Version Get Command for all Command Classes announced in NIF
  - Name: Default Node Name reported by NodeNaming Report
  - Location Default Node Location of Controller reported by NodeNaming Report
  - AppVersion: Application Version reported by ManufacturerSpecific Report
  - ManufacturerSpecific: Values report by ManufacturerSpecific Report
  - SpecificDeviceClass: Specific Device Class reported
  - GenericDeviceClass: Generic Device Class reported
- Command class 0x73 (Powerlevel) Timeout: Seconds until PowerLevel test will time out
- Command class 0x73 (Powerlevel) MaxFrames: Number of Test-Frames sent out per power level
- Command class 0x84 (Wakeup) WakeupInterval: Default Wakeup Interval
- Command class 0x2C (Scene Activation) Max Scenes: Maximum number of Scenes supported
- Command class 0x2D (Scene Activation) Max Scenes: Maximum number of Scenes supported
- Command class 0x75 (Protection) Mode: Default Protection Mode
- Command class 0x31 (SensorMultilevel) Fahrenheit: Flag what temperature scale is used
- Command class 0x27 (SwitchAll) Mode: default Switch All mode
- Command class 0x60 (Multichannel) ChannelsNumber: max number of channels emulated
- Command class 0x60 (Multichannel) GenericDeviceClass: Generic Device Class of simulated Channel
- Command class 0x60 (Multichannel) SpecificDeviceClass: Specific Device Class of simulated Channel

11.4 Subfolder /htdocs

This folder is the root file system of the built-in webserver. It contains the JavaScripts and HTMP pages of the Demo User Interface.

11.5 Subfolder /libs

This folder contains the binary libraries of the z-way server.

11.6 Subfolder /libzway-dev

This folder contains the C Header Files for the C API. The logic behind the C API is similar to the JSON API.

11.7 Subfolder /translations

This subfolder contains several XML files to translate numerical of the Z-Wave protocol into human readable Strings.

- AEC.xml: Advanced Energy Frame Work descriptions
- Alarms.xml: Alarm types
- DeviceClasses.xml: Device Classes of the Z-Wave
- RulesEvents.xml: different Event types
- SDKIds.xml: the System Development Kit revision numbers.
- Scales.xml. scales of Sensor Multilevel and Meter Command Class
- ThermostatModes.xml: different thermostat modes
- VendorIds.xml: The manufacturers of the Z-Wave devices.

11.8 File /z-way-server

This is the executable of Z-Way

12 Localization of Z-Way

Z-Way has three language specific data storages that need to be updated in order to add new language support.

- The Web UI: In htdocs/js there is a one language translation file per language, e.g. language.en.js. A new file needs to be generated with the two character language code (e.g. ru for Russian, ro for Romanian) and all strings on the right part of semicolon (:) on each lines need to be translated
- The Pepper-One Device Database: Z-Way takes all association group descriptions, device background info, configuration parameter and value descriptions from this database
- The folder /translations of Z-Way contains language specific IDs for sensor value scales, vendor information.

13 Command Class Reference

Command Classes are groups of wireless commands that allow using certain functions of a Z-wave device. In Z-Way each Z-Wave device has a data holder entry for each command class supported. During the inclusion and interview of the device the command class structure is instantiated in the data holder and filled with certain data. Command Class commands change values of the corresponding data holder structure. The follow list shows the public commands of the command classes supported with their parameters and the data holder objects changed.

The subsequent list shows the most important data holder values but the full set of data values can always be accesses using the demo UI. In expert mode there is a list of command classes with a simplified User Interface plus a link that visualizes all data holder elements.

13.1 Command Class Alarm Sensor (0x9c/156)

The Alarm Sensor Command Class can be used to realize Sensor Alarms.

Command Class values in data holder:

- 'type': A field with the type id as index what holds the information about the alarm sensors

Command Alarm Sensor Get

Syntax: Get(type = -1, successCallback = NULL, failureCallback = NULL)

Description: Requests the status of the alarm sensor of type 'type'

Parameter type: Alarm type to get. -1 means get all types

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "type id" is updated with sub variables srcId,sensorState,sensorTime

13.2 Command Class Association (0x85/133)

The association command class allows to manage the association groups and the nodeIDs in the association groups.

Command Class values in data holder:

- groups: number of association groups
- group number: array of association groups with the child values: max (max number of nodes allowed) and nodes as array of nodes in the association group

Command Association Get

Syntax: Get(groupId = 0, successCallback = NULL, failureCallback = NULL)

Description: Send Association Get

Parameter groupId: Group Id (from 1 to 255), 0 requests all groups

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder value "nodes" in association group object is updated

Command Association Set

Syntax: Set(groupId, includeNode, successCallback = NULL, failureCallback = NULL)

Description: Send Association Set (Add)

Parameter groupId: Group Id (from 1 to 255)

Parameter includeNode: Node to be added to the group

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder value "nodes" in association group object is updated

Command Association Remove

Syntax: Remove(groupId, excludeNode, successCallback = NULL, failureCallback = NULL)

Description: Send Association Remove

Parameter groupId: Group Id (from 1 to 255)

Parameter excludeNode: Node to be removed from the group

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder value "nodes" in association group object is updated

13.3 Command Class Basic (0x20/32)

The Basic Command Class is the wild card command class. Almost all Z-Wave devices support this command class but they interpret the command class commands in different ways. A Thermostat will handle a Basic Set Command in a different way than a Dimmer but both accept the Basic Set command and act.

Command Class values in data holder:

- level: generic switching level of the device controlled

Command Basic Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send Basic Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

Command Basic Set

Syntax: Set(value, successCallback = NULL, failureCallback = NULL)

Description: Send Basic Set

Parameter value: Value

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

13.4 Command Class Battery (0x80/128)

The battery command class allows monitoring the battery charging level of a device.
Command Class values in data holder:

- last: last battery charging level (0100
- history: an array of charging levels and UNIX time stamps, can be used to predict next time to change battery
- lastChange: UNIX time stamp of last battery change (if recognized)

Command Battery Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send Battery Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "last" is updated, in case "last" has changed, "history" or "lastChange" may be updates

13.5 Command Class Clock (0x81/129)

The clock Command Class allows to sync the internal clock for timer dependent application such as thermostats with schedules.

Command Class Clock Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send Clock Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Clock Set

Syntax: Set(successCallback = NULL, failureCallback = NULL)

Description: Send Clock Set

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.6 Command Class Configuration(0x70/112)

The configuration command class is used to set certian configuration valeus that change the behavior of the device. Z-Wave requires that every device works out of the box with out further configuration. However different configuration value significantly enhance the value a device.

Configuration parameters are identified by a 8 bit parameter number and a value that can be 1, 2 or even 4 byte long. Z-Wave does not provide any information about the configuration values by wireless commands. User have to look into the device manual to learn about configuration parameters. The Device Description Record, incoprotated by Z-Way gives information about valid parameters and the meaning of the values to be set.

Command Class values in data holder:

: Parameter value with child values size (1,2,4 byte) and value

Command Configuration Get

Syntax: Get(parameter, successCallback = NULL, failureCallback = NULL)

Description: Send Configuration Get

Parameter parameter: Parameter number (from 1 to 255)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder with parameter number is updated or created when no available

Command Configuration Set

Syntax: Set(parameter, value, size = 0, successCallback = NULL, failureCallback = NULL)

Description: Send Configuration Set

Parameter parameter: Parameter number (from 1 to 255)

Parameter value: Value to be sent (negative and positive values are accepted, but will be stripped to size)

Parameter size: Size of the value (1, 2 or 4 bytes). Use 0 to guess from previously reported value if any. 0 means use size previously obtained Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder with parameter number is updated

Command Configuration SetDefault

Syntax: SetDefault(parameter, successCallback = NULL, failureCallback = NULL)

Description: Send Configuration SetDefault

Parameter parameter: Parameter number to be set to device default

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.7 Command Class DoorLock (0x62/98)

The door lock command class allows to operate an electronic door lock
Command Class values in data holder:

- mode: general operating mode of lock
- insideMode: for inside handle
- outsideMode: for outside handle
- lockMinutes: setup value fo timeout
- lockSeconds: setup value fo timeout
- condition:
- insideState: state of inside handle

- outsideState: state of outside handle
- timeoutMinutes: time to timeout mode
- timeoutSeconds: time to timeout mode
- opType"));

Command Class DoorLock Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send DoorLock Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class DoorLock ConfigurationGet

Syntax: ConfigurationGet(successCallback = NULL, failureCallback = NULL)

Description: Send DoorLock Configuration Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class DoorLock Set

Syntax: Set(mode, successCallback = NULL, failureCallback = NULL)

Description: Send DoorLock Configuration Set

Parameter mode: Lock mode

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class DoorLock ConfigurationSet

Syntax: ConfigurationSet(opType, outsideState, insideState, lockMin, lockSec, successCallback = NULL, failureCallback = NULL)

Description: Send DoorLock Configuration Set

Parameter opType: Operation type

Parameter outsideState: State of outside door handle

Parameter insideState: State of inside door handle

Parameter lockMin: Lock after a specified time (minutes part)

Parameter lockSec: Lock after a specified time (seconds part)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.8 Command Class Door Lock Logging (0x4C/76)

The Door Lock Logging Command Class allows to receive reports about all successful and failed activities of the electronic door lock

Command Class values in data holder:

- log record: The data holder contains the log history

Command Door Lock Log LoggingGet

Syntax: LoggingGet(records,successCallback = NULL, failureCallback = NULL)

Description: Calls the log entries from the lock

Parameter records: max number of records

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder is updated

13.9 Command Class Indicator (0x87/135)

The indicator command class operates the indicator on the physical device if available. This can be used to identify a device or use the indicator for special purposes.

Command Class values in data holder:

- stat: The status of the indicator

Command Indicator Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Calls the indicator status from the device

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder stat is updated

Command Class Indicator Set

Syntax: Set(stat, successCallback = NULL, failureCallback = NULL)

Description: Send Indicator Set

Parameter stat: indicator status value

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.10 Command Class Meter (0x32/50)

The meter command class allows to read different kind of meters. Z-Wave differentiates different meter types and different meter scales. Please refer to the file /translations/scales.xml for details about possible meter types and values.

Command Class values in data holder:

- resettable: flag to indicate of the meter can be reset

typeId : One meter device can have different meters. Each meter object has the following child objects:

  - delta: difference between last and actual meter value.
  - previous: previous meter value (gotten with last GET request
  - rateType: meter rate type
  - scale: meter scale id
  - scaleString: string representation of meter scale. Refer to /translations/scales.xml for scale types.
  - sensorType: meter type id. Refer to /translations/scales.xml for types
  - sensorTypeString: string representation of sensor Type. Refer to /translations/scales.xml for type strings
  - val: The actual meter value

Command Meter Get

Syntax: Get(scale = -1, successCallback = NULL, failureCallback = NULL)

Description: Send Meter Get

Parameter scale: Desired scale. -1 for all scales

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder values of meter id are updated

Command Meter Reset

Syntax: Reset(successCallback = NULL, failureCallback = NULL)

Description: Send Meter Reset

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.11 Command Class Multichannel Association (0x8e/142)

This is an enhancement to the Association Command Class. The command class follows the same logic as the Association command class and has the same commands but accepts different instance values.

Command MultiChannelAssociation Get

Syntax: Get(groupId = 0, successCallback = NULL, failureCallback = NULL)

Description: Send MultiChannelAssociation Get

Parameter groupId: Group Id (from 1 to 255), 0 requests all groups

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command MultiChannelAssociation Set

Syntax: Set(groupId, includeNode, includeInstance, successCallback = NULL, failureCallback = NULL)

Description: Send MultiChannelAssociation Set (Add)

Parameter groupId: Group Id (from 1 to 255)

Parameter includeNode: Node to be added to the group

Parameter includeInstance: Instance of the node to be added to the group

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command MultiChannelAssociation Remove

Syntax: Remove(groupId, excludeNode, excludeInstance, successCallback = NULL, failureCallback = NULL)

Description: Send MultiChannelAssociation Remove

Parameter groupId: Group Id (from 1 to 255)

Parameter excludeNode: Node to be removed from the group

Parameter excludeInstance: Instance of the node to be removed from the group

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.12 Command Class NodeNaming (0x77/119)

The Node naming command class allows assigning a readable string for a name and a location to a physical device. The two strings are stored inside the device and can be called on request. There are no restrictions to the name except the maximum length of the string of 16 characters.

Command Class values in data holder:

- nodename: The name of the device
- location: the location of the device
- myname: The name of the Z-Way instance
- mylocation: the location of the Z-Way instance

Command NodeNaming Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send NodeNaming GetName and GetLocation

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "nodename" and "location" are updated

Command NodeNaming GetName

Syntax: GetName(successCallback = NULL, failureCallback = NULL)

Description: Send NodeNaming GetName

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "nodename" is updated

Command NodeNaming GetLocation

Syntax: GetLocation(successCallback = NULL, failureCallback = NULL)

Description: Send NodeNaming GetLocation

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "location" is updated

Command NodeNaming SetName

Syntax: SetName(name, successCallback = NULL, failureCallback = NULL)

Description: Send NodeNaming SetName

Parameter name: Value

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "nodename" is updated

Command NodeNaming SetLocation

Syntax: SetLocation(location, successCallback = NULL, failureCallback = NULL)

Description: Send NodeNaming SetLocation

Parameter location: Value

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "location" is updated

13.13 Command Class Protection (0x75/117)

This command class is used to disable local control of the device.

Command Class Protection Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send Protection Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Protection Set

Syntax: Set(state, rfState = 0, successCallback = NULL, failureCallback = NULL)

Description: Send Protection Set

Parameter state: Local control protection state

Parameter rfState: RF control protection state

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Protection ExclusiveGet

Syntax: ExclusiveGet(successCallback = NULL, failureCallback = NULL)

Description: Send Protection Exclusive Control Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Protection ExclusiveSet

Syntax: ExclusiveSet(controlNodeId, successCallback = NULL, failureCallback = NULL)

Description: Send Protection Exclusive Control Set

Parameter controlNodeId: Node Id to have exclusive control over destination node

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Protection TimeoutGet

Syntax: TimeoutGet(successCallback = NULL, failureCallback = NULL)

Description: Send Protection Timeout Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Protection TimeoutSet

Syntax: TimeoutSet(timeout, successCallback = NULL, failureCallback = NULL)

Description: Send Protection Timeout Set

Parameter timeout: Timeout in seconds. 0 is no timer set. -1 is infinite timeout. max value is 191 minute (11460 seconds). values above 1 minute are ...

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.14 Command Class SceneActivation(0x2B/43)

Command Class SceneActivation Set

Syntax: Set(sceneId, dimmingDuration = 0xff, successCallback = NULL, failureCallback = NULL)

Description: Send SceneActivation Set

Parameter sceneId: Scene Id

Parameter dimmingDuration: Dimming duration

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.15 Command Class SceneControllerConf (0x2d/45)

Command Class SceneControllerConf

Syntax: Get(group = 0, successCallback = NULL, failureCallback = NULL)

Description: Send SceneControllerConf Get

Parameter group: Group Id

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class SceneControllerConf Set

Syntax: Set(group, scene, duration = 0x0, successCallback = NULL, failureCallback = NULL)

Description: Send SceneControllerConf Set

Parameter group: Group Id

Parameter scene: Scene Id

Parameter duration: Duration

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.16 Command Class SceneActuatorConf (0x2C/44)

Command Class values in data holder:

- currentScene: the actual scene

Command Class SceneActuatorConf Get

Syntax: Get(scene = 0, successCallback = NULL, failureCallback = NULL)

Description: Send SceneActuatorConf Get

Parameter scene: Scene Id

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class SceneActuatorConf Set

Syntax: Set(scene, level, dimming = 0xff, override = TRUE, successCallback = NULL, failureCallback = NULL)

Description: Send SceneActuatorConf Set

Parameter scene: Scene Id

Parameter level: Level

Parameter dimming: Dimming

Parameter override: If false then the current settings in the device is associated with the Scene Id. If true then the Level value is used

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.17 Command Class ScheduleEntryLock (0x4e/78)

Controls access to a door lock based on times and intervals

Command Class ScheduleEntryLock Enable

Syntax: Enable(user, enable, successCallback = NULL, failureCallback = NULL)

Description: Send ScheduleEntryLock Enable(All)

Parameter user: User to enable/disable schedule for. 0 to enable/disable for all users

Parameter enable: TRUE to enable schedule, FALSE otherwise

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class ScheduleEntryLock WeekdayGet

Syntax: WeekdayGet(user, slot, successCallback = NULL, failureCallback = NULL)

Description: Send ScheduleEntryLock Weekday Get

Parameter user: User to get schedule for. 0 to get for all users

Parameter slot: Slot to get schedule for. 0 to get for all slots

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class ScheduleEntryLock WeekdaySet

Syntax: WeekdaySet(user, slot, dayOfWeek, startHour, startMinute, stopHour, stop-Minute, successCallback = NULL, failureCallback = NULL)

Description: Send ScheduleEntryLock Weekday Set

Parameter user: User to set schedule for

Parameter slot: Slot to set schedule for

Parameter dayOfWeek: Weekday number (0..6). 0 = Sunday. . 6 = Saturday

Parameter startHour: Hour when schedule starts (0..23)

Parameter startMinute: Minute when schedule starts (0..59)

Parameter stopHour: Hour when schedule stops (0..23)

Parameter stopMinute: Minute when schedule stops (0..59)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class ScheduleEntryLock YearGet

Syntax: YearGet(user, slot, successCallback = NULL, failureCallback = NULL)

Description: Send ScheduleEntryLock Year Get

Parameter user: User to enable/disable schedule for. 0 to get for all users

Parameter slot: Slot to get schedule for. 0 to get for all slots

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class ScheduleEntryLock YearSet

Syntax: YearSet(user, slot, startYear, startMonth, startDay, startHour, startMinute, stopYear, stopMonth, stopDay, stopHour, stopMinute, successCallba

Description: Send ScheduleEntryLock Year Set

Parameter user: User to set schedule for

Parameter slot: Slot to set schedule for

Parameter startYear: Year in current century when schedule starts (0..99)

Parameter startMonth: Month when schedule starts (1..12)

Parameter startDay: Day when schedule starts (1..31)

Parameter startHour: Hour when schedule starts (0..23)

Parameter startMinute: Minute when schedule starts (0..59)

Parameter stopYear: Year in current century when schedule stops (0..99)

Parameter stopMonth: Month when schedule stops (1..12)

Parameter stopDay: Day when schedule stops (1..31)

Parameter stopHour: Hour when schedule stops (0..23)

Parameter stopMinute: Minute when schedule stops (0..59)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.18 Command Class SensorBinary (0x30/48)

Command Class values in data holder:

- level: level of the binary sensor

Command SensorBinary Get

Syntax: Get(sensorType = 0, successCallback = NULL, failureCallback = NULL)

Description: Send SensorBinary Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter sensorType: Type of sensor to query information for. 0xFF to query information for the first available sensor type

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder value "level" is updated

13.19 Command Class Sensor Configuration (0x9e/158)

Allows to set a certain trigger level for a sensor to trigger

Command Class SensorConfiguration Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send SensorConfiguration Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class SensorConfiguration Set

Syntax: Set(mode, value, successCallback = NULL, failureCallback = NULL)

Description: Send SensorConfiguration Set

Parameter mode: Value set mode

Parameter value: Value

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.20 Command Class Sensor Multilevel (0x31/49)

The sensor multilevel command class allows to read different kind of sensor. Z-Wave differentiates different sensor types and different scales of this sensor. Please refer to the file /translations/scales.xml for details about possible sensor types and values.

Command Class values in data holder:

typeId : One sensor device can have different sensor. Each sensor object has the following child objects:

- scale: sensor scale id
- scaleString: string representation of sensor scale. Refer to /translations/scales.xml for scale types.
- sensorType: sensor type id. Refer to /translations/scales.xml for types
- sensorTypeString: string representation of sensor Type. Refer to /translations/scales.xml for type strings
- val: The actual sensor value

Command SensorMultilevel Get

Syntax: Get(sensorType = -1, successCallback = NULL, failureCallback = NULL)

Description: Send SensorMultilevel Get

Parameter sensorType: Type of sensor to be requested. -1 means all sensor types supported by the device

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder values of sensorIds are updated

13.21 Command Class Switch All (0x27/39)

This command class controls the behavior of a actuator on Switch all commands. It can accept, both on and off, only on, only off or nothing.
Command Class values in data holder:

- mode: the current acceptance mode

Command SwitchAll Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send SwitchAll Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder for "mode" is updated

Command SwitchAll Set

Syntax: Set(mode, successCallback = NULL, failureCallback = NULL)

Description: Send SwitchAll Set

Parameter mode: SwitchAll Mode: see definitions below

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder for "mode" is updated

Command SwitchAll SetOn

Syntax: SetOn(successCallback = NULL, failureCallback = NULL)

Description: Send SwitchAll Set On

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command SwitchAll SetOff

Syntax: SetOff(successCallback = NULL, failureCallback = NULL)

Description: Send SwitchAll Set Off

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.22 Command Class SwitchBinary(0x25/37)

The Switch Binary Command Class is used to control all actuators with simple binary (on/off) switching functions, primarily electrical switches.
Command Class values in data holder:

- Level: the level of the remotely controlled device
- mylevel: the level of the switch multilevel emulation of Z-Way

Command SwitchBinary Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send SwitchBinary Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder ”level” is updated

Command SwitchBinary Set

Syntax: Set(value, successCallback = NULL, failureCallback = NULL)

Description: Send SwitchBinary Set

Parameter value: Value

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

13.23 Command Class SwitchMultilevel (0x26/38)

The Switch Multilevel Command Class is used to control all actuators with multilevel switching functions, primarily Dimmers and Motor Controlling devices.
Command Class values in data holder:

- Level: the level of the remotely controlled device
- mylevel: the level of the switch multilevel emulation of Z-Way

Command SwitchMultilevel Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send SwitchMultilevel Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

Command SwitchMultilevel Set

Syntax: Set(level, duration = 0xff, successCallback = NULL, failureCallback = NULL)

Description: Send SwitchMultilevel Set

Parameter level: Level to be set

Parameter duration: Duration of change:. 0 instantly. 1-127 in seconds. 128-254 in minutes mapped to 1-127 (value 128 is 1 minute). 255 use device factory default

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

Command Class SwitchMultilevel StartLevelChange

Syntax: StartLevelChange(dir, duration = 0xff, ignoreStartLevel = TRUE, startLevel = 50, indec = 0, step = 0xff, successCallback = NULL, failureCallback = NULL)

Description: Send SwitchMultilevel StartLevelChange

Parameter dir: Direction of change: 0 to incrase, 1 to decrase

Parameter duration: Duration of change:. 0 instantly. 0x01-0x7f in seconds. 0x80-0xfe in minutes mapped to 1-127 (value 0x80=128 is 1 minute). 0xff us

Parameter ignoreStartLevel: If set to True, device will ignore start level value and will use it's curent value

Parameter startLevel: Start level to change from

Parameter indec: Increment/decrement type for step

Parameter step: Step to be used in level change in percentage. 0-99 mapped to 1-100%. 0xff uses device factory default

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

Command SwitchMultilevel StopLevelChange

Syntax: StopLevelChange(successCallback = NULL, failureCallback = NULL)

Description: Send SwitchMultilevel StopLevelChange

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "level" is updated

13.24 Command Class ThermostatFanMode(0x44/68)

Allows to control the Thermostat Fan
Command Class values in data holder:

- mode: fan mode

Command ThermostatFanMode Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatFanMode Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "mode" is updated

Command ThermostatFanMode Set

Syntax: Set(mode, successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatFanMode Set

Parameter mode: new fan mode

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "mode" is updated

13.25 Command Class ThermostatFanState(0x45/69)

Allows to control the Thermostat Fan
Command Class values in data holder:

- state: fan state

Command ThermostatFanState Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatFanState Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "state" is updated

13.26 Command Class ThermostatMode (0x40/64)

This command class allows to switch a heating/cooling actuator in different modes. During interview the mode mask is requested and the dat objects are create accordingly.

Command Class values in data holder:

- modemask: contains the modemask with bit to identify the different modes of the thermostat
- mode: the actual mode

modeID : list of all allowed modes with string representation.

Command ThermostatMode Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatMode Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command ThermostatMode Set

Syntax: Set(mode, successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatMode Set

Parameter mode: Thermostat Mode

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.27 Command Class ThermostatOperatingState (0x42/66)

This command class allows to determine the operating state of the thermostat

Command Class values in data holder:

- state: operating state

Command ThermostatOperatingState LoggingGet

Syntax:LoggingGet(successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatOperatingState LoggingGet

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.28 Command Class ThermostatSetPoint (0x43/67)

This command class allows to set a certain setpoint to a thermostat. The command class can be applied to different kind of thermostats (heating, cooling, ...), hence it has various modes.

Command Class values in data holder:

- modemask: contains the modemask with bit to identify the different modes of the thermostat

modeID : data object for each mode with the following child objects

  - modeName: contains the modemask with bit to identify the different modes of the thermostat
  - precision: data object for each mode with the following child objects
  - scale: scale id of the thermostat value
  - scaleString: string representation of the scale id
  - setVal
  - size: size of setpoint value in bte
  - val

Command ThermostatSetPoint Get

Syntax: Get(mode = -1, successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatSetPoint Get

Parameter mode: Thermostat Mode, -1 requests for all modes

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder for "mode" is updated

Command ThermostatSetPoint Set

Syntax: Set(mode, value, successCallback = NULL, failureCallback = NULL)

Description: Send ThermostatSetPoint Set

Parameter mode: Thermostat Mode

Parameter value: temperature

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder for "mode" is updated

13.29 Command Class UserCode (0x63/99)

Command Class UserCode Get

Syntax: Get(user = -1, successCallback = NULL, failureCallback = NULL)

Description: Send UserCode Get

Parameter user: User index to get code for (1 ... maxUsers). -1 to get codes for all users

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class UserCode Set

Syntax: Set(user, code, status = 0, successCallback = NULL, failureCallback = NULL)

Description: Send UserCode Set

Parameter user: User index to set code for (1...maxUsers) 0 means set for all users

Parameter code: Code to set (4...10 characters long)

Parameter status: Code status to set

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.30 Command Class Time (0x8a/138)

Command Class Time TimeGet

Syntax: TimeGet(successCallback = NULL, failureCallback = NULL)

Description: Send Time TimeGet

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Time DateGet

Syntax: DateGet(successCallback = NULL, failureCallback = NULL)

Description: Send Time DateGet

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class Time OffsetGet

Syntax: OffsetGet(successCallback = NULL, failureCallback = NULL)

Description: Send Time TimeOffsetGet

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.31 Command Class TimeParameters (0x8b/139)

Command Class TimeParameters Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send TimeParameters Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Command Class TimeParameters Set

Syntax: Set(successCallback = NULL, failureCallback = NULL)

Description: Send TimeParameters Set

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

13.32 Command Class Wakeup (0x84/132)

The wakeup command class handles the wakeup behavior of devices with wakeup interval
Command Class values in data holder:

- default: default wakeup interval (constant), only filled if device support Wakeup Command Class Version 2
- interval: wakeup interval in seconds
- lastSleep: UNIX time stamp of last sleep() command sent
- lastWakeup: UNIX time stamp of last wakeup notification() received
- max: maximum accepted wakeup interval (constant), only filled if device support Wakeup Command Class Version 2
- min: min. allowed wakeup interval (constant), only filled if device support Wakeup Command Class Version 2
- nodeId: Node ID of the device that will receive the wakeup notification of this device
- step: step size of wakeup interval setting allows (constant), only filled if device support Wakeup Command Class Version 2

Command Wakeup Get

Syntax: Get(successCallback = NULL, failureCallback = NULL)

Description: Send Wakeup Get

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "interval" is updated

Command Wakeup Sleep

Syntax: Sleep(successCallback = NULL, failureCallback = NULL)

Description: Send Wakeup NoMoreInformation (Sleep)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Return: data holder "lastsleep" is updated

Command Wakeup Set

Syntax: Set(interval, notificationNodeId, successCallback = NULL, failureCallback = NULL)

Description: Send Wakeup Set

Parameter interval: Wakeup interval in seconds

Parameter notificationNodeId: Node Id to be notified about wakeup

Return: data holder ”interval” and ”nodeId” is updated

14 Function Class Reference

Function Class GetSerialAPICapabilities

Syntax: GetSerialAPICapabilities(successCallback = NULL, failureCallback = None)

Description: Request Serial API capabilities

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SerialAPISetTimeouts

Syntax: SerialAPISetTimeouts(ackTimeout, byteTimeout, successCallback = NULL, failureCallback = NULL)

Description: Set Serial API timeouts

Parameter ackTimeout: Time for the stick to wait for ACK (in 10ms units)

Parameter byteTimeout: Time for the stick to assemble a full packet (in 10ms units)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SerialAPIGetInitData

Syntax: SerialAPIGetInitData(successCallback = NULL, failureCallback = NULL)

Description: Request initial information about devices in network

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SerialAPIApplicationNodeInfo

Syntax: SerialAPIApplicationNodeInfo(listening, optional, flirs1000, flirs250, genericClass, specificClass, nif, successCallback = NULL, failureCallback = NULL)

Description: Set controller node information

Parameter listening: Listening flag

Parameter optional: Optional flag (set if device supports more CCs than described as mandatory for it's Device Type)

Parameter flirs1000: FLiRS 1000 flag (hardware have to be based on FLiRS library to support it)

Parameter flirs250: FLiRS 250 flag (hardware have to be based on FLiRS library to support it)

Parameter genericClass: Generic Device Type

Parameter specificClass: Specific Device Type

Parameter nif: New NIF

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class WatchDogStart

Syntax: WatchDogStart(successCallback = NULL, failureCallback = NULL)

Description: Start WatchDog

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class WatchDogStop

Syntax: WatchDogStop(successCallback = NULL, failureCallback = NULL)

Description: Stop WatchDog

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class GetHomeId

Syntax: GetHomeId(successCallback = NULL, failureCallback = NULL)

Description: Request Home Id and controller Node Id

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class GetControllerCapabilities

Syntax: GetControllerCapabilities(successCallback = NULL, failureCallback = NULL)

Description: Request controller capabilities (primary role, SUC/SIS availability)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class GetVersion

Syntax: GetVersion(successCallback = NULL, failureCallback = NULL)

Description: Request controller hardware version

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class GetSUCNodeId

Syntax: GetSUCNodeId(successCallback = NULL, failureCallback = NULL)

Description: Request SUC Node Id

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class EnableSUC

Syntax: EnableSUC(enable, sis, successCallback = NULL, failureCallback = NULL)

Description: Enable or disable SUC/SIS functionality of the controller

Parameter enable: True to enable functionality, False to disable

Parameter sis: True to enable SIS functionality, False to enable SUC only

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SetSUCNodeId

Syntax: SetSUCNodeId(nodeId, enable, sis, successCallback = NULL, failureCallback = None)

Description: Assign new SUC/SIS or disable existing

Parameter nodeId: Node Id to be assigned/disabled as SUC/SIS

Parameter enable: True to enable, False to disable

Parameter sis: True to assign SIS role, False to enable SUC role only

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class MemoryGetByte

Syntax: MemoryGetByte(offset, successCallback = NULL, failureCallback = NULL)

Description: Read single byte from EEPROM

Parameter offset: Offset in application memory in EEPROM

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class MemoryGetBuffer

Syntax: MemoryGetBuffer(offset, length, successCallback = NULL, failureCallback = NULL)

Description: Read multiple bytes from EEPROM

Parameter offset: Offset in application memory in EEPROM

Parameter length: Number of byte to be read

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class MemoryPutByte

Syntax: MemoryPutByte(offset, data, successCallback = NULL, failureCallback = NULL)

Description: Write single byte to EEPROM

Parameter offset: Offset in application memory in EEPROM

Parameter data: Byte to be written

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class MemoryPutBuffer

Syntax: MemoryPutBuffer(offset, data, successCallback = NULL, failureCallback = NULL)

Description: Write multiple bytes to EEPROM

Parameter offset: Offset in application memory in EEPROM

Parameter data: Bytes to be written

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class IsFailedNode

Syntax: IsFailedNode(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Checks if node is failed

Parameter nodeId: Node Id to be checked

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SendDataAbort

Syntax: SendDataAbort(successCallback = NULL, failureCallback = NULL)

Description: Abort send data. Note that this function works unpredictably in multi callback environment !

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SerialAPISoftReset

Syntax: SerialAPISoftReset(successCallback = NULL, failureCallback = NULL)

Description: Soft reset. Restarts Z-Wave chip

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SendData

Syntax: SendData(nodeId, data, description = NULL, successCallback = NULL, failureCallback = NULL)

Description: Send data. Packets are sent in AUTO_ROUTE mode with EXPLRER_FRAME enabled for listening devices (ignored if not supported by the hardware [based on 5.0x branch])

Parameter nodeId: Destination Node Id (NODE_BROADCAST to send non-routed broadcast packet)

Parameter data: Paket payload

Parameter description: Packet description for queue inspector and logging

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class GetNodeProtocolInfo

Syntax: GetNodeProtocolInfo(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Get node protocol info

Parameter nodeId: Node Id of the device in question

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class GetRoutingTableLine

Syntax: GetRoutingTableLine(nodeId, removeBad = False, removeRepeaters = False, successCallback = NULL, failureCallback = NULL)

Description: Get routing table line

Parameter nodeId: Node Id of the device in question

Parameter removeBad: Exclude failed nodes from the listing

Parameter removeRepeaters: Exclude repeater nodes from the listing

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class AssignReturnRoute

Syntax: AssignReturnRoute(nodeId, destId, successCallback = NULL, failureCallback = NULL)

Description: Assign return route to specified node. Get Serial API capabilities

Parameter nodeId: Node Id of the device that have to store new route

Parameter destId: Destination Node Id of the route

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class AssignSUCReturnRoute

Syntax: AssignSUCReturnRoute(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Assign return route to SUC

Parameter nodeId: Node Id of the device that have to store route to SUC

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class DeleteReturnRoute

Syntax: DeleteReturnRoute(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Delete return route

Parameter nodeId: Node Id of the device that have to delete all assigned return routes

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class DeleteSUCReturnRoute

Syntax: DeleteSUCReturnRoute(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Delete return route to SUC

Parameter nodeId: Node Id of the device that have to delete route to SUC

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SetDefault

Syntax: SetDefault(successCallback = NULL, failureCallback = NULL)

Description: Reset the controller. Note: this function will delete ALL data from the Z-Wave chip and restore it to factory default !. Sticks based on 4.5x and 6.x SDKs will also generate a new Home Id.

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SendSUCNodeId

Syntax: SendSUCNodeId(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Send SUC Node Id. Informs portable and static controllers about new or deleted SUC/SIS

Parameter nodeId: Node Id of the device that have to be informed about new or deleted SIC/SIS

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SendNodeInformation

Syntax: SendNodeInformation(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Send NIF of the stick

Parameter nodeId: Destination Node Id (NODE_BROADCAST to send non-routed broadcast packet)

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class RequestNodeInformation

Syntax: RequestNodeInformation(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Request NIF of a device

Parameter nodeId: Node Id to be requested for a NIF

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class RemoveFailedNode

Syntax: RemoveFailedNode(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Remove failed node from network. Before removing SDK will check that the device is really unreachable

Parameter nodeId: Node Id to be removed from network

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class ReplaceFailedNode

Syntax: ReplaceFailedNode(nodeId, successCallback = None, failureCallback = None)

Description: Replace failed node with a new one. Be aware that a failed node can be replaced by a node of another type. This can lead to problems. Always request device NIF and force re-interview after successful replace process.

Parameter nodeId: Node Id to be replaced by new one

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class RequestNetworkUpdate

Syntax: RequestNetworkUpdate(successCallback = NULL, failureCallback = NULL)

Description: Request network topology update from SUC/SIS. Note that this process may also fail due more than 64 changes from last sync. In this case a re-inclusion of the controller (self) is required.

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class RequestNodeNeighbourUpdate

Syntax: RequestNodeNeighbourUpdate(nodeId, successCallback = NULL, failureCallback = NULL)

Description: Request neighbours update for specific node

Parameter nodeId: Node Id to be requested for its neighbours

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class SetLearnMode

Syntax: SetLearnMode(startStop, successCallback = NULL, failureCallback = NULL)

Description: Set/stop Learn mode

Parameter startStop: Start Learn mode if True, stop if False

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class AddNodeToNetwork

Syntax: AddNodeToNetwork(startStop, highPower = True, successCallback = NULL, failureCallback = NULL)

Description: Start/stop Inclusion of a new node. Available on primary and inclusion controllers

Parameter startStop: Start inclusion mode if True, stop if False

Parameter highPower: Use full power during this operation if True. On False use low power mode.

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class RemoveNodeFromNetwork

Syntax: RemoveNodeFromNetwork(startStop, highPower = False, successCallback = NULL, failureCallback = NULL)

Description: Start/stop exclusion of a node. Note that this function can be used to exclude a device from previous network before including in ours. Available on primary and inclusion controllers

Parameter startStop: Start exclusion mode if True, stop if False

Parameter highPower: Use full power during this operation if True. On False use low power mode.

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class ControllerChange

Syntax: ControllerChange(startStop, highPower = False, successCallback = NULL, failureCallback = NULL)

Description: Set new primary controller (also known as Controller Shift). Same as Inclusion, but the newly included device will get the role of primary.. Available only on primary controller.

Parameter startStop: Start controller shift mode if True, stop if False

Parameter highPower: Use full power during this operation if True. On False use low power mode.

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class CreateNewPrimary

Syntax: CreateNewPrimary(startStop, successCallback = NULL, failureCallback = NULL)

Description: Create new primary controller by SUC controller. Same as Inclusion, but the newly included device will get the role of primary.. Available only on SUC.. Be careful not to create two primary controllers! This can lead to network malfunction!

Parameter startStop: Start create new primary mode if True, stop if False

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

Function Class ZMEFreqChange

Syntax: ZMEFreqChange(freq, successCallback = NULL, failureCallback = NULL)

Description: Change Z-Wave.Me Z-Stick 4 frequency. This function is specific for Z-Wave.Me hardware

Parameter freq: 0 for EU, 1 for RU

Parameter successCallback: Custom function to be called on function success. NULL if callback is not needed

Parameter failureCallback: Custom function to be called on function failure. NULL if callback is not needed

15 Z-Way Data Model Reference

15.1 (root)

- Description: the root of the data model
- Syntax: X with X as child object
- Child objects
  - setTimeout(function() , timeout): equivalent to the setTimeout function of Javascript, used for scripting
  - setInterval(function() , timeout): equivalent to the setInterval function of Javascript, used for scripting
  - clearTimeout(function() , timeout): equivalent to the clearTimeout function of Javascript, used for scripting
  - clearInterval(function() , timeout): equivalent to the clearInterval function of Javascript, used for scripting
  - debugPrint(string): used for debug prints
  - executeJS(string): executes javascript code in the string (like eval)
  - executeFile(string): execute js code from file
  - loadObject(object_name): returns JSON object from storage
  - saveObject(object_name, object): saves JSON object to filesystem storage
  - system(command): executes an external command in shell and returns [ret_code, output]. Command should start with one of the lines in http_root/storage/.syscommands

15.2 zway

- Description: zway is the Z-Way part of the object tree
- Syntax: zway.X with X as child object
- Child objects
  - controller: controller object, see below for details
  - devices: devices list, see below for details
  - version: Z-Way.JS version
  - isRunning(): Check if Z-Way is running
  - isIdle(): Check if Z-Way is idle (no pending packets to send)
  - discover(): Start Z-Way discovery process
  - stop() : Stop Z-Way
  - InspectQueue() : Returns list of pending jobs in the queue.
    - item: [timeout, flags, nodeId, description, progress, payload]
    - flags: [send count, wait wakeup, wait security, done, wait ACK, got ACK, wait response, got response, wait callback, got callback]

- ProcessPendingCallbacks(): Process pending callbacks (result of setTimeout/setInterval or functions called via HTTP JSON API)
- bind(function, bitmask): Bind function to be called on change of devices list/instances list/command classes list
- unbind(function) : Unbind function previously bind with bind()
- all function classes in 14 are also methods of this data object

15.3 controller

You can access the data elements of "controller" in the demo user interface in menu "for experts + Controller Info"

- Description: Controller object
- Syntax: controller.X with X as child object
- Child objects
  - data: Data tree of the controller
    - APIVersion: Version of the Serial API
    - SDK: System development kit version of the Transceiver firmware
    - SISPresent: flase if SUIS is available
    - SUCNodeId: Node ID of SUC if present
    - ZWVersion: ZWave Version (firmware)
    - ZWaveChip: The name of the Z-Wave transceiver chip
    - ZWlibMajor / ZWlibMinor: library version
    - capabilities: array of function class ids
    - controllerstate: flag to show inclusion mode etc
    - countJobs: shall job be counted
    - curSerialAPIAckTimeout10ms: timing parameter of serial interface
    - curSerialAPIBytetimeout10ms: timing parameter of serial interface
    - homeId:the home id of the controller
    - isinOtherNetworks: flag to show if controller is real primary if in other network
    - isPrimary: flag to show if controller is primary
    - isRealprimary: flag to show if controller can be primary
    - isSUC: is SUC present
    - lastExcludedDevice: node ID of last excluded device
    - lastIncludedDevice: node ID of last included device
    - libType library basis type
    - manufacturerIS / manufacturerProductId / manufacturerProductTypeId: ids to identify the transceiver hardware
    - memoryGetAddress
    - memoryGetData
    - nodeId: own node ID
    - nonManagementJobs: number of non man. jobs

    * oldSerialAPIAckTimeout10ms: default timing parameter of serial interface
    * oldSerialAPIBytetimeout10ms: default timing parameter of serial interface
    * softwareRevisonDate: written by compiler
    * softwareRevisionID: written by compiler
    * vendor: string of hardware vendor
  - AddNodeToNetwork(mode): Reference to zway.AddNodeToNetwork()
  - RemoveNodeFromNetwork(mode): Reference to zway.RemoveNodeFromNetwork()
  - ControllerChange(mode): Reference to zway.ControllerChange()
  - GetSUCNodeId(mode): Reference to zway.GetSUCNodeId()
  - SetSUCNodeId(nodeId): Assign SUC role to a device
  - SetSISNodeId(nodeId): Assign SIS role to a device
  - DisableSUCNodeId(nodeId): Revoke SUC/SIS role from a device
  - SendNodeInformation(nodeId): Reference to zway.SendNodeInformation()

15.4 Devices

The devices object contains the array of the device objects. Each device in the network - including the controller itself - has a device object in Z-Way.

- Description: list of devices
- Syntax: X with X as child object
- Child objects

  m : Device object
  - length: Length of the list
  - SaveData(): Save Z-Way Z-Wave data for hot start on next run (in config/zddx/¡homeID¿-DevicesData.xml)

15.5 Device

The data object can be accesses in the demo UI in advanced mode of "Configuration"

- Description: the device object
- Syntax: device[n].X with X as child object
- Child objects
  - id: (node) Id of the device
  - Data: Data tree of the device
    * SDK: SDK used in the device
    * ZDDXMLFile: file of the Devcie Description Record
    * ZWLib: Z-Wave library used
    * ZWProtocolMajor / ZWProtocolMinor: Z-Wave protocol version
    * applicationMajor / ApplicationMinor: Application Version of devices firmware

    * basicType: basic Z-Wave device class
    * beam: wake up beam required
    * countFailed: statistics of failed packets sent (from start of process)
    * countSuccess: statistics of sucessful packets sent (from start of process)
    * deviceTypeString:
    * genericType: generic Z-Wave device class
    * infoProtocolSpecific
    * isAwake
    * isListening
    * isFailed
    * isRouting
    * isVirtual:
    * keeAwake: flag is device need to be kept awake
    * lastRecevied: timestamp of last packet received
    * lastSend: timestamp of last sent operation
    * ManufacturerId / manufacturerProductID / manufacturerProductTypeId: ids used to identify the device
    * neightbours: list of neighbour nodes
    * nodeInfoFrame: nodeinformation frame in bytes
    * option: flag if optional command classes are present
    * queueLength: length of device specific send queue
    * sensor1000: flag if device is FLIRS with 1000 ms wakeup
    * sensor250: flag if device is FLIRS with 250 ms wakeup
    * specificType: specific Z-Wave device class
  - instances: Instances list of the device
  - RequestNodeInformation(): Request NIF
  - RequestNodeNeighbourUpdate(): Request routes update
  - InterviewForce(): Purge all command classes and start interview based on device's NIF
  - RemoveFailedNode(): Remove this node as failed. Device should be marked as failed to remove it with this function.
  - SendNoOperation(): Ping the device with empty packet
  - LoadXMLFile(file): Load new Z-Wave Device Description XML file. See http://pepper1.net/zwavedb/
  - GuestXML(): Return the list of all known Z-Wave Device Description XML files with match score. [score, file name, brand name, product name, photo]
  - WakeupQueue(): Pretend the device is awake and try to send packets

15.6 Instances

Each device may have multiple instances (similar functions like switches, same type sensors, ...) If only one instance is present the id of this instance is 0. Command classes are located in instances only-

- Description: list of instances

- Syntax: device[n].instance[m].X with X as child object
- Child objects

  m : instance object
  - length: Length of the list
  - commandClasses: list of command classes of this instance. In case there is only one instance, this is equivalent to the list of command classes of the device. For details see below.
  - Data: data object of instance
    - dynamic: flag if instance is dynamic
    - genericType: generic Z-Wave device class of instance
    - specificType: specific Z-Wave device class of instance

15.7 CommandClass

This is the command class object. It contains public methods and public data elements that are described in chapter 13

- Description: Command Class Implementation
- Syntax: device[n].instance[m].commandclass[id].X with X as child object
- Child objects
  - id: Id of the Command Class of the instance of the device
  - data: Data tree of the Command Class
    - interviewCounter: number of attempts left until interview is terminate even if not successful
    - interviewDone: flag if interview of the command class is finished
    - security: flag if command class is operated under special security mode
    - version: version of the command class implemented
    - supported: flag if CC is supported or only controlled
    - commandclass data: Command Class specific data - see chapter 13 for details.
  - name: Command Class name
  - Method: Command Class method - see chapter 13 for details.

15.8 Data

This is the description of the data object.

General note: Z-Way objects and it's decendents are NOT simple JS objects, but native JS objects, that does not allow object modification.

- name: Name of data tree element
- updated: Update time
- invalidated: Invalidate time

- valueOf(): Returns value of the object (can be omitted to get object value)
- invalidate(): Invalidate data value (mark is as not valid anymore)
- bind(function (type[, arg]) ..., [arg, [watchChildren=false]]): Bind function to a change of data tree element of its descendants
- unbind(function): Unbind function bind previously with bind()