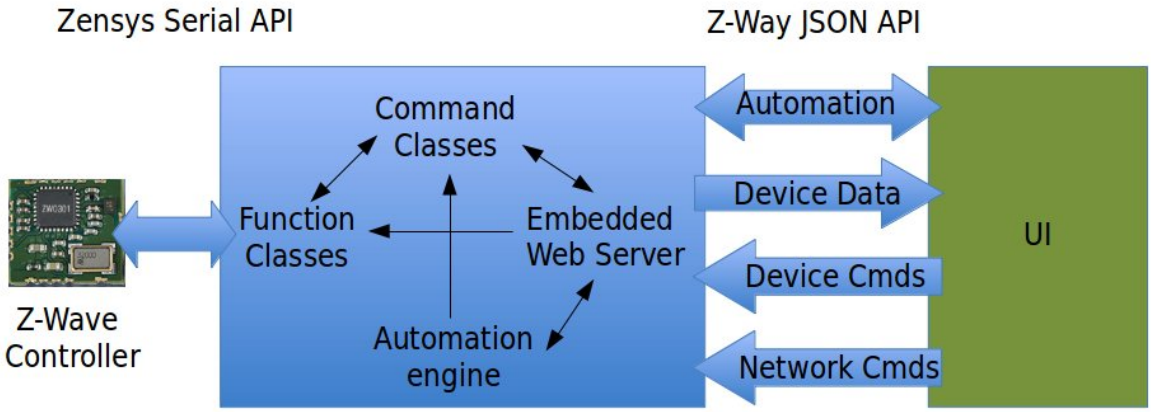
Z-Way API

Version Dec 2012

Z-Way uses HTTP API to communicate with User Interfaces (UI) or external programs. Using Z-Way API one can access data structures in Z-Way and execute per devices and you can handle network management functions.

Structure of Z-Way

Z-Way consists of internal Web Server, Command Classes data and functions, Function Classes functions and Automation engine data and functions. Z-Wave controller finally understands only Function Classes commands and hence Z-Way translates every command into Function Classes function calls.



Z-Wave basis

In Z-Wave notation **Function Classes** are commands to talk with the Z-Wave chip (they are coded into Sigma Designs Serial API functions to be transferred over serial interface to the Z-Wave transceiver). For example, SendData, AddNodeToNetwork, RemoveNodeFromNetwork are Function Classes commands.

Each **Command Class** (CC) defines device functionality with all commands accepted and responded by the device. Command Class commands are sent to devices using SendData or SendDataMulti Function Classes. Command Classes specifies how to talk between Z-Wave devices and guarantees interoperability.

For example, the Basic Command Class is defined as Get, Set and Report commands and is mandatory to be supported on most of devices to allow basic interoperability between devices. The meaning of the value sent by Set depends on device type (**Device Class** in Z-Wave notation).

The SwitchBinary Command Class also defines Get, Set, Report, but it's meaning is different: in Set value 0 = Off, 1-99, 255 = On. All other values are ignored. This command is supposed to be used on devices having only two states (like relay).

List of supported Command Class for a devices are reported in **Node Information Frame** (NIF).

Each device might consist of several **instances**. Most of devices have only one instance (instance 0), but some devices have several instances to allow user to e.g. control different relays in one device or gather information from different sensors in one device. For example, temperature and humidity sensor will have two instances 1 and 2 and default instance 0 referring to the device itself. Each instance might have it's own list of Command Classes reported by the device and stored in data holder of the instance.

JSON (JavaScript Object Notation) basis

Z-Way uses JSON to transfer data from backend server to user frontend (UI). In JSON data are represented by their name and value as String in hierarchical form. To build hierarchy associative arrays are used. (Note that in JavaScript any type derived from Object is in fact an associative array!). JSON is very popular for exchanging data over web interfaces but also used for inter process communication.

Advantages:

- Well structured, open data structures
- Easy to read and to parse

Disadvantages:

- String representation takes a lot of memory. Hence, compression use is suggested

More information about JSON can be found at http://en.wikipedia.org/wiki/JSON

Execute functions in Z-Way

To execute a command in Z-Way, send an HTTP **POST** request to a specific URL of the embedded webserver.

For example: http://localhost:8083/ZWaveAPI/Run/controller.DoSomething()

Here is the list of important function:

- controller.AddNodeToNetwork(M) — start (M=1) / stop (M=0) inclusion process
- controller.RemoveNodeFromNetwork(M) — start (M=1) / stop (M=0) exclusion process
- controller.SetLearnMode(M) — start (M=1) / stop (M=0) learn mode process
- controller.SetDefault() — reset Z-Wave controller to default. NB: will completely destroy all stored data about your network!

It is also possible to execute commands related to devices, its instances or to command class of a particular instance.
For example:

- devices[nodeId].DoSomething(parameter)
- devices[nodeId].instances[instanceId].DoSomething(param)
- devices[nodeId].instances[instanceId].commandClasses[ccId].DoSomething(param)
- devices[nodeId].instances[instanceId].commandClasses.SwitchBinary.DoSomething(param)

Each Command Class in Z-Wave has a set of functions. They are used to access the functionality of this CC of for internal maintenance. The internal maintenance functions of CC are hidden by Z-Way. All public commands can be accessed and tested using the *Experts Commands* section in the *Experts* mode of the Z-Way Default Installers Web UI.

To simplify generation of Command Classes functionality UI, each Command Class has a list of all exported function and description of available parameters. Functions are split into four types (read / change • user / configuration data):

- userGet — sends request (usually Get command) to the device
- userSet — sends command changing state of the device (usually Set command)
- configGet — sends request (usually Get command) to the device to get it's configuration
- configSet — sends command changing state of the device (usually Set command)

Each type stores list of commands. Each command stores a list of parameters, while each parameter stores tree describing it's UI as

```
label: "[string]", [type]: [type description]
```

List of all types used in CC descriptions:

- fix — fixed value, can be used only inside *enumof* to implement radio buttons
- range — value from *min* to *max*, can also be shifted by *shift* value
- node — dropdown list of nodes in network, only one node can be selected
- string — string value
- enumof — defines list of exclusive parameter UI (radio button with child UI), can contain as child*fix*, *range*

Get data from Z-Way

All Z-Way data is transferred as one single JSON object (let's call it *D* for simplicity). It is also possible to get updates of this object since last update. An associative array (again a JSON object) of the following type is returned back from server:

```
{     "[path in D object]": [updated subtree],     "[path in D object]": [updated subtree],
...     updateTime: [current timestamp] }
```

To keep last update time, *D* object always contains an *updateTime* variable containing timestamp.

To get updates, send a request to the following URL:

- /ZWaveAPI/Data/0 — Get all data (updates since 1970)
- /ZWaveAPI/Data/134500000 — Get updates since 134500000 (Unix timestamp)

The data object *D.controller* contains all network/controller related data in hierarchical format.
The data object *D.devices* contains all device related data in hierarchical format

- *D.devices[nodeID].data.nodeId* contains the nodeID of the device
- *D.devices[nodeID].instances[instanceID].data.specificType* contains specific device type fo the instance instanceID of the device nodeID.
- *D.devices[nodeID].instances[instanceID].SensorMultilevel.data.val* contains value reported by a multilevel sensor

Each Data element such as *D.devices[nodeID].data.nodeId* is an object with the following child 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
- updated — boolean value indicating if the device is update or invalid (in time between issuing a Get and receving a Report)

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

Handling of updates coming from Z-Way

A good design of UI would be one linking UI objects (label, textbox, slider, ...) with a a certain path in *D*object. So, any update of a subtree linked to UI would update the UI too. This is called binding in Microsoft terms (present for example in WPF). Z-Way contains a library called jQuery.triggerPath (extention of jQuery), that allows making such links between objects in *D* object and HTML DOM objects. Use

```
jQuery.triggerPath.init(D);
```

during web application initialization and 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 *D*, 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:

```
$('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 2-10 seconds):

```
// Call this function periodically or manually to get updates. // Parameter sync allows to make
request synchronous or asynchronous
function getDataUpdate(sync) {
  $.postJSON('/ZWaveAPI/Data/' + D.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 = D;
           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);
    };
};
```

Localization of Z-Wav

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.

For more information visit http://www.z-wave.me or mail to info@zwave-me