Stretch miniREST v2 API

Introduction

The *Plugwise Stretch miniRest API* is a simplified and somewhat limited API for controlling the Plugwise Stretch and the connected Plugwise modules. Goal is to reduce the required coding on the client side and still offer all of the functionalities for daily operation of the Stretch in a easy to use way. This document is intended for developers who have a good understanding of the Plugwise products, XML and HTTP requests (i.e. AJAX).

License

Support

The *Plugwise Stretch miniRest API* is not supported by the Plugwise Helpdesk. However, if you have a question or find a bug, feel free to send an email to the helpdesk@plugwise.com.

Disclaimer

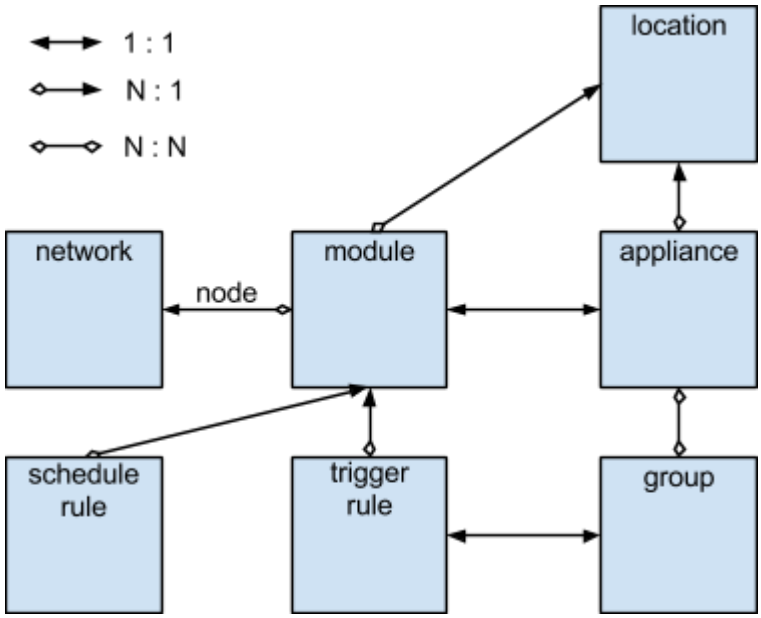
The *Plugwise Stretch miniRest API* is considered to be a beta release. Although Plugwise will do its best to maintain backwards compatibility, specifications might change without prior notice.

Plugwise B.V.
Wattstraat 56
2171 TR Sassenheim
The Netherlands
T: +31(0) 252 43 30 74
F: +31(0) 252 43 30 79
E: info@plugwise.com

Contents

Datamodel

The data model is based on the Source' data model, rather than the Stretch's internal data model.



- On the Stretch, 'location' is the equivalent of the 'room' object of Source
- On the Stretch, 'rule' replaces both the 'trigger' and 'schedule' object of Source
- In Source a group can have 0 or more triggers, on the Stretch, a Group can have 0 or 1
- On the Stretch for each trigger rule without a group, a switching group is created
- In Source a schedule is linked to an appliance, on the Stretch a schedule rule is linked to a module
- A Plugwise network consists of nodes rather than modules. A node is always a module, but a module does not need to be a node, i.e. a smart meter module.

Supported HTTP methods

GET	retrieving objects
POST	creating and updating objects and relations
DELETE	deleting objects and relations

Each request must send the HTTP header

```
Content-Type: text/xml
```

POST and DELETE can be mimicked in a GET by using `_method_=POST` resp. `_method_=DELETE` in the URL. The data to POST must be URL-encoded and passed with parameter `_body_`.

Testing

A page to view and test API calls can be found on the Stretch at the url /miniresttest.html

Objects

Objects are represented by XML nodes. You can GET objects by class or GET all objects by one request using

```
GET: /minirest/allobjects
RESULT:
       <restobjects>
       ...
       </restobjects>
```

Objects in the output are sorted by name.

Note: In the examples in this document, most objects ids are shorted to the form ‘xxxx...xx’, in real life you should always use the full 32 characters of an id.

Network

```
GET: /minirest/networks
RESULT:
    <network id="4485...27">
      <type>pwzigbee</type>
      <name>000D6F00004EEA32</name>
      <created_date>2012-08-30T21:15:11+02:00</created_date>
      <last_seen_date>2012-08-31T19:12:51+02:00</last_seen_date>
      <modified_date>2012-08-31T19:12:51+02:00</modified_date>
      <scan_for_nodes>off</scan_for_nodes>
      <nodes>
        <node id="0971...24">
          <mac_address>000D6F00004EEA32</mac_address>
          <type>0</type>
        </node>
        <node id="c261...83">
          <mac_address>000D6F0000138B49</mac_address>
          <type>1</type>
        </node>
        <node id="4ce8...08">
          <mac_address>000D6F00001C7C53</mac_address>
          <type>2</type>
        </node>
       ...
      </nodes>
      <nodes_available>
      <node>
```

```
      <mac_address>000D6F000076D0E4</mac_address>
      <last_seen_date>2012-09-02T18:18:15+02:00</last_seen_date>
    </node>
    </nodes_available>
    <nodes_to_add>
      <node><mac_address>000D6F0001A403C5</mac_address></node>
      <node><mac_address>000D6F0001A4035D</mac_address></node>
    </nodes_to_add>
    <nodes_to_delete>
      <node><mac_address>000D6F0000C3CD81</mac_address></node>
    </nodes_to_delete>
  </network>
```

A network node is always a module, the module id is the same as the node id, but a module does not need to be a node, for example a smart meter module.

Module

Type	ZigBee	Product name
0	MC	Stick, Built-in in Stretch
1	NC	Circle+, Stealth+, Sting+
2	NR	Circle, Stealth, Sting
3	SSW	Switch
5	SSN	Sense
6	SSC	Scan

```
GET: /minirest/modules
RESULT:
    <module id="762a...32">
      <mac_address>000D6F0000B84FEB</mac_address>
      <name>00B84FEB</name>
      <type>3</type>
      <hardware_version>6539-0800-2904</hardware_version>
      <firmware_version>2012-06-27</firmware_version>
      <created_date>2012-08-30T21:15:55+02:00</created_date>
      <modified_date>2012-08-31T19:56:37+00:00</modified_date>
      <last_seen_date>2012-08-31T19:56:37+00:00</last_seen_date>
      <network id="4485...27"/>
    </module>
```

A NC and NR have the additional elements

```
<power_state>on</power_state>
<current_power_usage>0</current_power_usage>
<last_known_measurement_date>2012-08-30T21:16:39+00:00</last_known_measurement_date>
```

A SSN has the additional elements

```
<current_temperature>26.00</current_temperature>
<current_humidity>35.60</current_humidity>
<last_known_measurement_date>2012-08-31T19:56:44+00:00</last_known_measurement_date>
```

For filters, a module can not just be identified by its id, but also by its mac address or a (significant) end part of the mac address.
For example. A module with id ‘d0381cad05144de0a50698d35b1b27b8‘ and mac address ‘000D6F0000C3CD81’ can be filtered by

```
GET: /minirest/modules;id=d0381cad05144de0a50698d35b1b27b8
GET: /minirest/modules;id=000D6F0000C3CD81
GET: /minirest/modules;id=0C3CD81
etc.
```

To get the appliance this module is linked to

```
GET: /minirest/appliances;module=d0381cad05144de0a50698d35b1b27b8
GET: /minirest/appliances;module=0C3CD81
etc.
```

It is advised not to use shortened mac addresses of less than 6 character, to prevent matching the wrong module.

Appliance

```
GET: /minirest/appliances
RESULT:
    <appliance id="c2f3...a0">
      <name>The Appliance</name>
      <type>radio</type>
      <created_date>2012-08-31T08:20:23+00:00</created_date>
      <modified_date>2012-08-31T20:14:53+00:00</modified_date>
      <last_seen_date>2012-08-31T20:14:53+00:00</last_seen_date>
      <power_state>off</power_state>
      <current_power_usage>0</current_power_usage>
      <do_not_switch_off>false</do_not_switch_off>
    <last_known_measurement_date>2012-08-31T18:01:38+00:00</last_known_measurement_date>
      <module id="fcfa...d3"/>
      <location id="0996...76"/>
      <groups>
        <group id="f239...f8"/>
      </groups>
    </appliance>
```

The elements ‘groups’ and ‘location’ only exist if one or more relations exists.

Group

```
GET: /minirest/groups
RESULT:
```

```
<group id="f239...f8">
  <name>A Group</name>
  <type>switching</type>
  <created_date>2012-08-31T08:20:44+00:00</created_date>
  <modified_date>2012-08-31T20:14:53+00:00</modified_date>
  <appliances>
    <appliance id="bcf6...bb"/>
    <appliance id="c2f3...a0"/>
  </appliances>
</group>
```

The element ‘appliances’ only exists if one or more relations exists.

Location

```
GET: /minirest/locations
RESULT:
    <location id="0996...76">
      <name>A Location</name>
      <type>livingroom</type>
      <created_date>2012-08-31T20:13:16+00:00</created_date>
      <modified_date>2012-08-31T20:14:53+00:00</modified_date>
      <appliances>
        <appliance id="c2f3...a0"/>
      </appliances>
      <modules>
        <module id="abdf...ae"/>
      </modules>
    </location>
```

The element ‘appliances’ only exists if one or more relations exists.

Rule

A rule object is automatically created for each trigger on a module (like buttons on a Switch)

Type	Product name
button_left	Switch, left button
button_right	Switch, right button
thermo_trigger	Sense, temperature
hygro_trigger	Sense, humidity
motion_trigger	Scan, movement
schedule	Circle (+), schedule

```
GET: /minirest/rules
RESULT:
    <rule id="17e3...b6">
      <name>00B84FEB_button_left</name>
      <type>button_left</type>
      <created_date>2012-08-30T21:15:55+00:00</created_date>
      <modified_date>2012-08-31T19:56:37+00:00</modified_date>
      <last_seen_date>2012-08-31T19:56:37+00:00</last_seen_date>
      <modules>
        <module id="762a...32"/>
      </modules>
      <group id="6aae...43"/>
    </rule>
```

A thermo of hygro trigger has the additional element

```
      <settings>
        <upwards boundary="0" action="off"/>
        <downwards boundary="0" action="off"/>
      </settings>
```

A motion trigger has the addition element

```
      <settings sensitivity="medium" daylight_override="off" delay="15"/>
```

A rule is changed by posting the changes:

```
POST: /minirest/rules
RESULT:
    <rule id="17e3...b6">
      <settings sensitivity="high" daylight_override="off" delay="5"/>
    </rule>
```

A switching group is created and linked for and to each trigger rule. This group can be updated or replaced and even deleted by the user, but as soon as a rule is no longer linked to a group, a new group is created and linked automatically.

Schedule rules are not automatically created. You can create as many schedules as you need, but a module (like a Circle) can have only 1 schedule rule.

The format for a schedule rule is

```
    <rule id="ec68...09">
      <name>First schedule</name>
      <type>schedule</type>
      <created_date>2013-12-19T19:59:34+01:00</created_date>
      <modified_date>2013-12-23T13:15:24+01:00</modified_date>
      <modules>
        <module id="32ec...8e"/>
        <module id="df77...54"/>
      </modules>
      <settings>
        <edge day="mo" time="06:45" value="on"/>
```

```
      <edge day="mo" time="22:00" value="off"/>
      <edge day="tu" time="06:45" value="on"/>
      <edge day="tu" time="22:00" value="10"/>
      <edge day="we" time="06:45" value="on"/>
      <edge day="we" time="22:00" value="10"/>
      <edge day="th" time="06:45" value="off"/>
      <edge day="th" time="22:00" value="19"/>
      <edge day="fr" time="06:45" value="on"/>
      <edge day="fr" time="22:00" value="10"/>
      <edge day="sa" time="06:45" value="on"/>
      <edge day="sa" time="22:00" value="100"/>
      <edge day="su" time="06:45" value="on"/>
      <edge day="su" time="22:00" value="1990"/>
    </settings>
  </rule>
```

Unlike Source, a schedule on the Stretch consist of setpoints. Day codes are ‘mo’, ‘tu’, ‘we’, ‘th’, ‘fr’, ‘sa’, ‘su’. The ‘time’ value must be rounded to the quarters of an hour. Allowed values for ‘value’ are ‘on’, ‘off’ and standby values in Watts.

Enumerations

A list of supported appliance, groups, and location types can be retrieved with

```
GET: /minirest/enumerations
RESULT:
    <enumerations>
      <group>
        <application/>
        <report/>
    ...
      </group>
      <location>
        <bathroom/>
    ...
    </enumerations>
```

Creating and updating objects

The miniREST API automatically creates and links an appliance object for every power metering module (NC and NRs) and a switching group for every rule. It does not automatically delete these created objects, when the module or rule is deleted.
Objects of classes appliance, group, location and Schedule rule can be created by the user. Objects of classes network, module and rule (except Schedule rule) are created automatically by the Stretch.

On creation only the name and type should be specified, other properties must be set by updating the object.

To create an object, an XML is sent as the body of an HTTP-POST, with one or more elements of the object type. The result is a list with the created objects.

Example:

```
POST: /minirest/appliances
BODY:
    <appliances>
      <appliance>
        <name>The Appliance</name>
        <type>radio</type>
      </appliance>
    </appliances>
```

Updating is done in the same manner, except that in the body the object's element has an 'id' attribute with the uuid of the object.

Example:

```
POST: /minirest/appliances
BODY:
    <appliance id="12c1...59">
      <type>dryer</type>
      <do_not_switch_off>true</do_not_switch_off>
    </appliance>
```

Deleting objects

Only objects created by a user can be deleted by a user; networks, modules and trigger rules cannot be deleted.

To delete an object, the id of the object is sent in the URL of a HTTP-DELETE

Example:

```
DELETE:/minirest/appliances;id=12c1...59
```

Linking objects

As shown in the data model, objects can have relations. Infact most objects have relations with one or more other objects. For example a module can be linked to an appliance.

To link (relate) an object to one or more other objects, the id of the object is specified in the URL, together with the plural of the type of the objects than will be linked, followed by the IDs of the objects to link.

```
POST: /minirest/appliances;id=c11f...96/modules;id=758d...96
BODY:
```

- In a URL a relation is always plural even if the relation can only be singular.
- If a new relations conflict with an existing relation (i.e. the appliance is already linked to a module), the existing relation is removed.

Unlinking objects

To remove the link (relation) between 2 objects, a HTTP-DELETE is used.

Example

```
DELETE /minirest/appliances;id=c11f...96/modules;id=758d...96
```

Cascading relations and filters

When querying objects using HTTP-GET, relations can be cascaded in the URL and filters can be used to retrieve specific sets of objects. The syntax is:

```
<type+s>[[;<attribute><operator><value>[,...]][...][/relation+s]][...]
```

Supported operators are:

```
=, !=, >, >=, <, <=
```

Examples:

The groups of a specific appliance

```
/minirest/appliances;id=bcf6...bb/groups
```

All the modules that are linked to any appliance:

```
/minirest/appliances/modules
```

Locations that have a appliance linked to any module:

```
/minirest/modules/appliances/locations
```

Locations that have a radio:

```
/minirest/appliances;type=radio/locations
```

Bedrooms that have television or a radio

```
/minirest/appliances;type=radio,tv/locations;type=bedroom
```

Radios and televisions placed in bedrooms

```
/minirest/locations;type=bedroom/appliances;type=radio,tv
```

Radios and televisions placed in bedrooms that currently use more than 50 Watts

```
/minirest/locations;type=bedroom/appliances;type=radio,tv;current_power_usage>50
```

Date and time values

Date values are returned in ISO 8601 format and should also be sent in that format. The format is

```
YYYY-MM-DD[<T|<space>>hh:mm[:ss][<+|->th:tm]]
```

Examples of valid dates:

```
2012-09-01T13:19:14+02:00
2012-09-01 13:19:14+02:00
2012-09-01 13:19+02:00
2012-09-01T13:19
2012-09-01
```

If the time part of a date is omitted in a filter, then only the date part of a property is used in that filter.

Using ‘now’ and ‘today’

A date can be compared to the current date and time by using ‘now’ as value. Using ‘`now(<offset>)`’ will use the current date and time added with the specified offset in seconds, which can be negative. For example to get the modules that were seen in the last minute:

```
/minirest/modules;last_seen_date>now(-60)
```

Another special value for date filters is ‘`today`’, which takes the date part of the current date and time. Using ‘`today(<days>[,<minutes>])`’ will add ‘days’ and optionally ‘minutes’ to 00:00 am today. If ‘minutes’ is omitted, only the resulting date part is used. With ‘minutes’, even if it is ‘0’, the whole date and time is used. Thus ‘`today`’ and ‘`today(0)`’ both result in ‘`2012-09-01`’ while ‘`today(0,0)`’ results in ‘`2012-09-01T00:00:00+02:00`’.

All modules that have been seen today:

```
/minirest/modules;last_seen_date=today
```

All Senses that have been seen after 10:00 am today

```
/minirest/modules;type=5;last_seen_date>today(0,600)
```

Modules: Linking an appliance

There are 2 methods for linking a module to an appliance. The obvious method is to POST the module id to the appliance:

```
POST:   /minirest/appliances;id=a169...1d/modules;id=92d4...26
```

The second method is to POST the appliance to the module. With this method you can create or update an appliance and link it to the module in one request:

```
POST:   /minirest/modules;id=92d4...26/appliances
BODY:
        <appliances>
```

```
        <appliance>
        <name>Lamp front</name>
        <type>lamp</type>
        </appliance>
RESULT:
        <appliances>
        <appliance id="a169...1d">
        <name>Lamp front</name>
        <type>lamp</type>
        <created_date>2012-09-01T18:08:44+02:00</created_date>
        <last_seen_date>2012-09-01T18:10:08+02:00</last_seen_date>
        <power_state>off</power_state>
        <current_power_usage>0</current_power_usage>
        <do_not_switch_off>false</do_not_switch_off>
        <last_known_measurement_date></last_known_measurement_date>
        <module id="92d4...a926"/>
        </appliance>
        </appliances>
```

Switching: Appliances and groups

Appliances and groups can be switched on and off. This is done in a POST with an empty body.

```
POST /minirest/appliances;id=bcf6...bb/power_state=on
```

More than one id can be specified, each separated by a comma.

Switching: Modules

Modules can be switched on and off. This is done in a POST with an empty body.

```
POST /minirest/modules;filter/power_state=on
```

The result of *filter* must be a single module object. For instance

```
POST /minirest/modules;mac_address=1C7C91/power_state=off
```

Note: Unlike switching an appliance or group, this call will wait until the module has the new state or after a timeout of 5 seconds after which it will return the last known state. Check the ‘modified_date’ property to verify this.

Network: Scanning, adding and removing nodes

You can scan for, add and remove nodes (modules) from the Plugwise network of the Stretch. Linking the Stretch to a NC however is not yet supported.
To start scanning for available nodes and/or to allow new nodes in the ZigBee network, the NC of the network must be set to scanning:

```
POST /minirest/networks;id=c11f...96/scan_for_nodes=on
```

When finished scanning for or adding nodes, you must switch off scanning or it will have impact on the

performance of the ZigBee network.

```
POST /minirest/networks;id=c11f...96/scan_for_nodes=true
```

Like with modules, you can shorten a MAC address.

To add one or more nodes to a network, POST the node's MAC address to the network:

```
POST: /minirest/networks;id=c11f...96/nodes;mac_address=099278D,0B84FEB
```

Until a node is accepted by the NC and has joined the ZigBee network, the node is listed in the `nodes_to_add` element of the network.

To remove one or more nodes from a network, use DELETE:

```
DELETE: /minirest/networks;id=c11f...96/nodes;mac_address=000D6F000099278D,0B84FEB
```

Until the node has left the ZigBee network and the registration is removed from the NC, the node is listed in the `nodes_to_delete` element of the network.

Examples

Example 1: Creating, linking, naming and switching an appliance

Suppose you have 2 lamps in your living room that you want to switch using the right button of a Plugwise Switch with id `00B84FEB`.
All you have to do is:

1. Create an appliance for each lamp
2. Link each appliance to the module the corresponding lamp is connected to
3. Add the appliances to the switching group of right button rule of the Switch

The first 2 steps can be combined to one request per module.

1&2. Create each appliances by posting it directly to its module

```
POST:   /minirest/module;id=92d4...26/appliances
BODY:
        <appliances>
        <appliance>
        <name>Lamp front</name>
        <type>lamp</type>
        </appliance>
        </appliances>
RESULT:
        <appliances>
        <appliance id="a169...1d">
        <name>Lamp front</name>
        <type>lamp</type>
        <created_date>2012-09-01T18:08:44+02:00</created_date>
        <last_seen_date>2012-09-01T18:10:08+02:00</last_seen_date>
        <power_state>off</power_state>
        <current_power_usage>0</current_power_usage>
        <do_not_switch_off>false</do_not_switch_off>
        <last_known_measurement_date></last_known_measurement_date>
        <module id="92d4...26"/>
        </appliance>
        </appliances>

POST:   /minirest/module;id=ea9c...8f/appliances
BODY:
        <appliances>
        <appliance>
        <name>Lamp back</name>
        <type>lamp</type>
        </appliance>
        </appliances>
RESULT:
        <appliances>
        <appliance id="41b9...84">
        <name>Lamp back</name>
```

```
<type>lamp</type>
<created_date>2012-09-01T18:08:44+02:00</created_date>
<last_seen_date>2012-09-01T18:11:50+02:00</last_seen_date>
<power_state>off</power_state>
<current_power_usage>0</current_power_usage>
<do_not_switch_off>false</do_not_switch_off>
<last_known_measurement_date></last_known_measurement_date>
<module id="ea9c...8f"/>
</appliance>
</appliances>
```

3. Add appliances to the switching group:

You can GET the correct group by the request:

```
GET:    /minirest/rules;module=00B84FEB;type=button_right/groups
RESULT:
        <groups>
        <group id="d2ea...1a">
        <name>Group_00B84FEB_button_right</name>
        <type>switching</type>
        <created_date>2012-09-01T17:14:06+02:00</created_date>
        <rule id="e892...29"/>
        </group>
        </groups>

POST:   /minirest/groups;id=d2ea...1a/appliances;id=a169...1d,41b9...84
RESULT:
        <groups>
        <group id="d2ea...1a">
        <name>Group_00B84FEB_button_right</name>
        <type>switching</type>
        <created_date>2012-09-01T17:14:06+02:00</created_date>
        <appliances>
        <appliance id="a169...1d"/>
        <appliance id="41b9...84"/>
        </appliances>
        <rule id="e892...29"/>
        </group>
        </groups>
```

After some seconds, the Stretch has programmed the modules to respond to the Switch and the lamps can be switched by the right button. You can also use a POST to switch the lamps:

```
POST: /minirest/groups;id=d2ea...1a/power_state=on
POST: /minirest/groups;id=d2ea...1a/power_state=off
```

Example 2: Adding and removing modules to and from a network

Only modules that are not yet linked to a network can be added to a network. To add a module to the

Plugwise ZigBee network you must know the MAC address of that module or you can ‘scan’ for available modules and add use MAC addresses with the risk that you add a module that is not yours, i.e. its plugged in in a neighbouring house.

The first step is to tell the Stretch to allow (new) modules in the network:

```
POST /minirest/networks;id=c11f...96/scan_for_nodes=on
```

Next you add the MAC address(es) of the module(s) to the network:

```
POST: /minirest/networks;id=c11f...96/nodes;mac_address=1A20C41
```

You can shorten the MAC addresses to a minimum of the last 6 digits, but we advise to use at least the last 7.

For knowing the (intermediate) result you call:

```
POST: /minirest/networks;id=c11f...96
RESULT:
<?xml version="1.0" encoding="iso-8859-1"?>
<networks count="1">
  <network id="c11f...96">
    <name>000D6F00004EEA32</name>
    <type>pwzigbee</type>
    <created_date>2012-11-02T10:40:24+00:00</created_date>
    <modified_date>2012-11-06T07:26:00+02:00</modified_date>
    <last_seen_date>2012-11-06T07:26:00+02:00</last_seen_date>
    <scan_for_nodes>on</scan_for_nodes>
    <nodes>
      <node id="b957...66">
        <mac_address>000D6F00004EEA32</mac_address>
        <type>0</type>
      </node>
      ...
      <node id="7197...6b">
        <mac_address>000D6F0000D3588E</mac_address>
        <type>2</type>
      </node>
    </nodes>
    <nodes_available>
      <node>
        <mac_address>000D6F0001A20DE1</mac_address>
        <last_seen_date>2012-11-06T04:25:45+02:00</last_seen_date>
      </node>
      ...
      <node>
        <mac_address>000D6F0001A20C41</mac_address>
        <last_seen_date>2012-11-06T04:23:32+02:00</last_seen_date>
      </node>
    </nodes_available>
    <nodes_to_add>
```

```
        <node>
          <mac_address>000D6F0001A403C5</mac_address>
        </node>
      </nodes_to_add>
      <nodes_to_remove/>
    </network>
  </networks>
```

As soon as the module is accepted in the network, it will be removed from the ‘nodes_to_add’ list appear in the ‘nodes’ list.
Available modules (modules that are currently not part of any network) will appear in the ‘nodes_available’ list.

To remove a module from the network you use:

```
DELETE: /minirest/networks;id=c11f...96/nodes;mac_address=000D6F0000D3588E
```

The module will be added to the ‘nodes_to _remove’ list, but will also remain in the ‘nodes’ list. As soon as the has been removed from the network it will disappear from both lists.

Note that to be removed a module has be online, so especially for SEDs it might take awhile before the module has been removed from the network.

Stretch discovery

The Plugwise Stretch supports mDNS (a.k.a. Apple Bonjour) and advertises the web server.
On Linux systems you can discover a Stretch on the local network with avahi.

```
avahi-browse -ptr _plugwise._tcp
```

Apart from mDNS the Stretch has it's own passive advertising service. The service listens for a specific multicast and replies to it. It does not send any info unless requested for.

Multicast address: 224.9.9.1
Multicast port: 4991

A stretch can reply to three different requests strings:

1) `mc=<mac address of internal Stick>`
Find the Stretch that has a Stick with that specific MAC address.
For example 'mc=000D6F00004EEA32'

2) `name=<host name>`
Find the Stretch with a specific hostname.
For example: 'name=stretch000020'

3) `plugwise stretch`
Find any Stretch on the network.

When receiving a matching request string on 224.9.9.1:4991, the Stretch will respond with an XML:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<stretch_info>
  <product>stretch</product>
  <firmware>firmware version</firmware>
  <hostname>hostname</firmware>
  <lan_ip>LAN IP (if active)</lan_ip>
  <wifi_ip>WiFI IP (if active)</wifi_ip>
</stretch_info>
```

For example:

```
<stretch_info>
  <product>stretch</product>
  <firmware>2.2.10</firmware>
  <hostname>stretch123b4d</hostname>
  <lan_ip>10.0.1.162</lan_ip>
  <wifi_ip/>
</stretch_info>
```