VORWEG GEHEN

Lemonbeat Application Layer Specification Draft

04 September 2015

DRAFT

This document contains the specification draft for the Lemonbeat Application Layer.

Version 1.8

Contents

Chapter 1

Introduction

The Lemonbeat Protocol is a Protocol aimed to provide a solution for low cost and low power connectivity for devices using batteries as a power source. The Protocol aims to solve the problems that exist in currently available protocols.

The problems in current protocols are among others:

- Missing or bad security
- Bad robustness
- Interference with other technology
- Requirements for a gateway centric system are not met
- Duty cycle limits
- Range
- Bad routing strategies
- Not standards based
- No good application layer

All layers and services in Lemonbeat from the data link layer and up are based on Internet standards. The application layer described in this document is self-descriptive and extendable and based on standard technology that is widely used in the internet today.

1.1 Document revision history

Ver.	Rev.	Description	Authors
1	0	Initial draft version Reviewed by	Andreas Madsen (Seluxit), Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Leni Lausdahl (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)
1	3	Updated value types and units. Updated network inclusion. Removed unused tags and attributes. Reviewed by	Andreas Madsen (Seluxit), Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)
1	4	Changed memory id's Renamed conditions to calculation Reviewed by	Andreas Madsen (Seluxit), Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)
1	5	Changed PHY header to better support forward error correction. Reviewed by	Andreas Madsen (Seluxit), Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)
1	6	Converted document to LaTeX Change PHY header for Forward Error Correction Updated services due to new XSD's Added status and configuration service Remove Partner Link service Reviewed by	Andreas Bomholtz (Seluxit) Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)
1	7	Updated the status service Reviewed by	Andreas Bomholtz (Seluxit) Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)
1	8	Added new Generic MAC options Added new special virtual values Reviewed by	Andreas Bomholtz (Seluxit) Daniel Lux (Seluxit), Henrik Sorensen (Seluxit), Morten Frederiksen (Seluxit) Michael Westermeier (RWE)

Chapter 2

Application

In Section 2.1 the application layer is described. In Section 2.2 the Lemonbeat Application Protocol is described and in the next section XML example for the protocol is presented. In the last section in this chapter, user stories for some scenarios and the xml solutions are presented.

2.1 Application layer

This chapter defines the different frames used in the application layer of the Lemonbeat protocol.

2.1.1 NTP

The Network Time Protocol is used to synchronize the clocks of Lemonbeat devices. Based on the synchronized clocks a Wake on radio enabled device must calculate the points in time where the receiver is in the RX on state. Devices that need to wake up Wake on radio devices also use the synchronized clocks to calculate the correct time for sending a wake up frame. NTP uses the UDP on port number 123. Lemonbeat devices must implement the NTP protocol as defined in RFC-5905.

2.1.1.1 NTP timestamps

Lemonbeat devices use the native NTP Timestamp format which is defined as in Table 2.1.

Data Size in Bits	Description
32	Seconds since start of era (era 0 starts 0h 1 January 1900 UTC, negative numbers are before this date)
32	Fraction of a second

Table 2.1: NTP Timestamp

2.1.1.2 NTP frame

The NTP frame is built up as in Table 2.2.

Size in Bits	Description
2	Leap Indicator
3	Version Number
3	Mode
8	Stratum
8	Poll max interval
8	Precision
32	Root delay
32	Root dispersion
32	Reference ID
64	Reference timestamp
64	Origin timestamp
64	Receive timestamp
64	Transmit timestamp
X	Extension Field 1
X	Extension Field 2
32	Key Identifier
128	Digest

Table 2.2: NTP Frame

In Lemonbeat all fields but the Extension Field 1, Extension Field 2, Key Identifier and Digest are used.

2.1.1.3 Clock synchronization algorithm

Lemonbeat devices must only synchronize time with devices that are in direct range, i.e. no routers are used when communicating with the other device is needed. When sending NTP frames, the CSMA-CA maximum wait time must be set to 0. The reception timestamp of a NTP frame must be set to the time when the Sync-word has been detected.

The round-trip delay is computed as in Equation 2.1, where t0 is the time of the request packet transmission, t1 is the time of the request packet reception, t2 is the time of the response packet transmission and t3 is the time of the response packet reception. t3 - t0 is the time elapsed on the client side between the emission of the request packet and

the reception of the response packet, while t2 - t1 is the time the server waited before sending the answer. The offset is given in Equation 2.2.

$$\delta = (t3 - t0) - (t2 - t1) \tag{2.1}$$

$$\theta = \frac{(t1 - t0) - (t2 - t3)}{2} \tag{2.2}$$

2.2 Protocol description

The application protocol is defined in XML to make the messages more human readable and easier to define the protocol messages. But XML is too big to use in an embedded application protocol, so it is compressed to be more usable in the embedded system. The XML messages are compressed using a method called Efficient XML Interchange (EXI) that converts XML into a binary event stream. The specification is defined by the W3C (`www.w3.org`) and the specification can be found at `http://www.w3.org/TR/2011/REC-exi-20110310/`. In this way an example XML document of 274 bytes can be compressed to be 4 bytes without loss of information. See XML 2.1 for an example of an XML message of 1253 Bytes which is compressed into an EXI message shown in XML 2.2 of 83 Bytes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="device_description.xsd">
  <device version="1">
    <device_description_report>
      <!-- Type -->
      <info number="1" type_id="1"/>
      <!-- Manufacturer -->
      <info hex="00112233445566778899AABB" type_id="2"/>
      <!-- Sgtin -->
      <info hex="AABBCCDDEEFF" type_id="3"/>
      <!-- Mac Address -->
      <info number="1" type_id="4"/>
      <!-- Hardware Version -->
      <info number="14" type_id="5"/>
      <!-- Bootloader Version -->
      <info number="1" type_id="6"/>
      <!-- Stack Version -->
      <info hex="00340080" type_id="7"/>
      <!-- Application Version -->
      <info number="500" type_id="8"/>
      <!-- Protocol -->
      <info number="10000" type_id="9"/>
      <!-- Product -->
      <info number="150" type_id="10"/>
      <!-- Included -->
      <info number="2" type_id="11"/>
      <!-- Name -->
      <info number="4" type_id="12"/>
      <!-- Radio Mode -->
      <info number="1" type_id="13"/>
      <!-- Wakeup Interval -->
      <info string="Device name" type_id="14"/>
    </device_description_report>
  </device>
</network>
```

XML 2.1: XML to EXI compression XML version

```
80 00 50 09 10 14 04 00 c0 01 12 23 34 45 56 67 78 89 9a ab b8 08 00 6a ab bc cd de ef f8 0c 10
14 10 10 e4 14 10 14 18 00 40 03 40 08 08 1c 1f 40 34 20 19 04 e4 24 19 60 14 28 10 24 2c 10 44
30 10 14 34 20 d4 46 57 66 96 36 52 06 e6 16 d6 50 71 68
```

XML 2.2: XML to EXI compression EXI version (values in hexadecimal notation)

2.2.1 Services

The applications listen on TCP and UDP ports, and depending on which port the message arrives at, it can be determined what kind of message it is. The ports are defined in Table 2.3. In a message there is a source and destination port. The destination post is the defined service port, and the source port can be a randomly selected port from 20128 to 20256 or one of the defined service ports. This port pair gives a connection between the sender and receiver. All answers will be sent back to the source port, so the sender knows that the incoming message is the answer. So when sending a value get message, then the destination port will be the value port and the source port will be a random port, because the message

that has been sent is a value message. The reply will be send back with the ports flipped, so that the source port is the value port.

The port the devices listens on start from port 20000. The ports can be compressed so port 20000 is mapped to port 0 and port 20256 will be 256 as defined in Table 2.3. In the next section the inclusion and exclusion of devices will be described.

Port number	Compressed port	Name
20000	0	Value
20001	1	Device Description
20002	2	Public Key
20003	3	Network Management
20004	4	Value description
20005	5	Service description
20006	6	Memory information
20007	7	Partner information
20008	8	Action
20009	9	Calculation
20010	10	Timer
20011	11	Calendar
20012	12	State machine
20013	13	Firmware update
20014	14	Channel Scan
20015	15	Status
20016	16	Configuration
...	...	

Table 2.3: List of port numbers

2.2.2 Network management

When a device A starts up and it is not included, it must start by sending out a device description message, signaling that device A wants to be included into a network. When a Network Controller receives a device description message, the Network Controller can decide if device A should be included or not. If the Network Controller decides to include device A, it needs the RSA public key of device A. The Network Controller can receive this key from the back end, or if device A supports it, the key can be retrieved from device A.

When the Network Controller has the public key of device A, it will generate a new unique AES key (Controller key) that will be used to encrypt the Network Key. The controller key and the encrypted network key is then added to a message along with a crc of them both. The message format can be seen in Table 2.4.

This message in then encrypted with the public key of Device A, so that only Device A with its private key can decrypt the message and obtain the network key. See Figure 2.1 for an illustration of the inclusion of device A.

Bytes	Description
0 - 15	A 16 byte AES controller key
16 - 31	A 16 byte AES network key AES encrypted with the controller key
32 - 33	A CRC 16 of the controller key and encrypted network key

Table 2.4: The format of the inclusion message

Alongside the Network Key, there is a number that tells the device how many bytes of its mac address it should use as its new address. The new address will for example be converted from `0x1234567890ab` to `0x90ab` if the address size is 2.

In the device when it receives the Network Key and the address size, it changes the value of its included property to 1. It also adds the compressed mac address to the list of address it listens on. Now device A is included and can communicate to the other devices in the network.

When device A needs to be excluded from the network, the Network Controller sends a message that sets the included property to 0. When device A receives this message it do factory reset, which clears all its values and parameters. Now the device is ready to be included again. See Figure 2.2 for an illustration of an exclusion of a device.

In the next sections there will be described how a device can be configured.

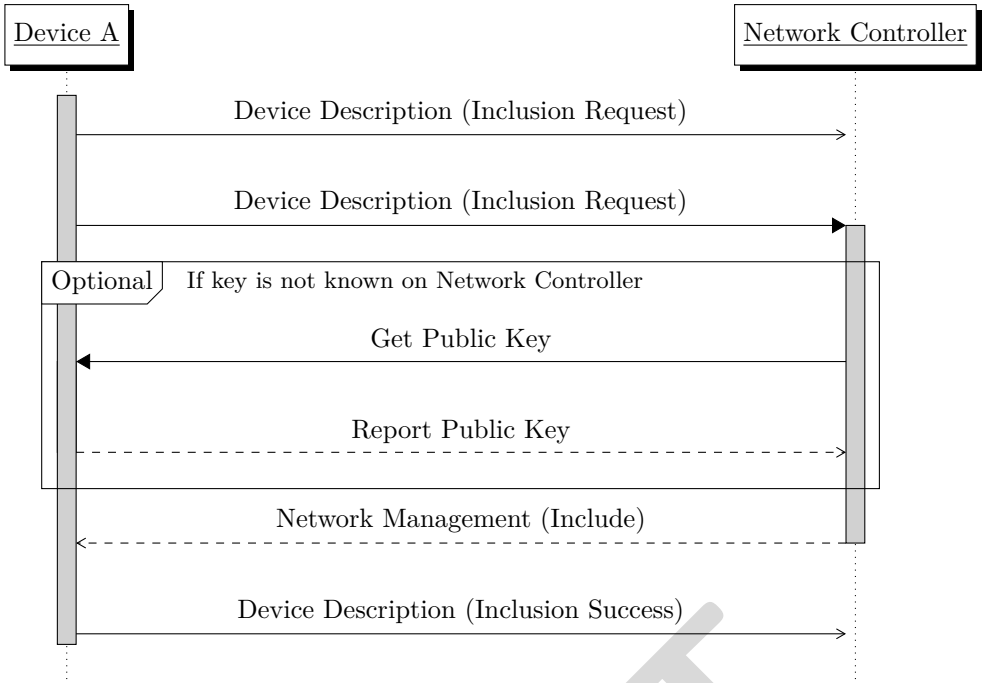

Figure 2.1: Inclusion

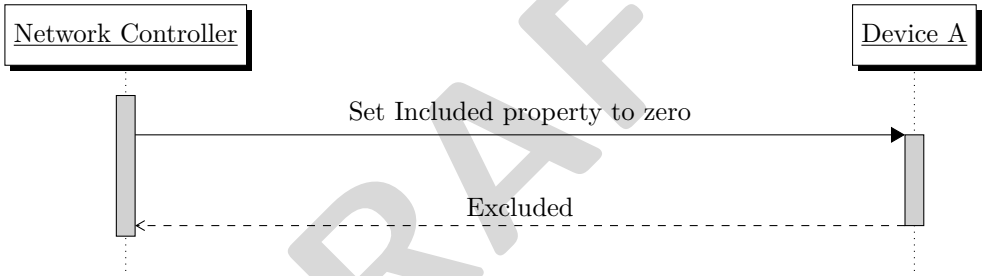

Figure 2.2: Exclusion

2.2.3 Device Configuration

Normally the behavior of a device is hard coded in the application of the device. Sometimes the device supports some configuration parameters to make small changes to the behavior. Often the behavior of the device does not meet future requirements of a user. If the complete behavior of a device could be configured, such a device would be more future proof. This goal can be obtained using state machines. These state machines can be constructed and tested on a PC and then be transferred to the device.

When a device is included into a network, only the default configuration, if any, is present on the device. If the device support configuration then the device can be configured. When ever a device has been configured, the new configuration has to be saved before it becomes active. The configuration service is used to handle the saving and clearing the configuration on the device. The configuration service is described in Section 2.2.3.8. The rest of the configuration services is described in the following sections.

2.2.3.1 Virtual Values

A device can support virtual values, which is values that can be configured externally. These values can be used as a variable when doing complex calculations and state machines. If a virtual value is added with a type that the device do not support, it will send a status report back.

There is special configurations for a virtual value, that has a special meaing for the stack. The different configurations is described in Table 2.5. All of these special virtual values must be configured with `persistent` set to zero.

Name	Type	Mode	Unit	Description
Stop Watch	TIME	R/W	ms	The value can be set to zero to reset the stop watch, and when ever it is read, it returns the elapsed time.
Timezone Offset	TIMEZONE_OFFSET	R/W	s	The value is used to read and set the current timezone offset.
Year	YEAR	R	y	The value is used to get the current year.
Month	MONTH	R	mo	The value is used to get the current month.
Day of Month	DAY_OF_MONTH	R	d	The value is used to get the current day of month.
Weekday	WEEKDAY	R	d	The value is used to get the current weekday.
Hour of Day	HOUR	R	h	The value is used to get the current hour of the day.
Minute of Hour	MINUTE	R	min	The value is used to get the current minute of the hour.

Table 2.5: Special Virtual Values

2.2.3.2 Partner Service

The partner service is used to map an address to a simple id. In this way it is simple to reference other devices by their address by only using the id. This partner id can be used in all the other services to get easy method to address other devices. A partner consistent of an address and information about how to communicate with the device. Only devices which is configuration as partners can talk to the device. If a multicast address is configurated as a partner, then the device will listen and accept messages send to that this multicast address.

The partner service also support grouping multiple partners into a group with a id. If the device sends to a group, the message will be send to all the partners in the group. If one or more of the partners in the group is a multicast address, then the message is first send to all multicast address in the group.

2.2.3.3 Timer Service

The timer service is used for executing action with a delay. The delay is specified in milliseconds. Further more, a condition can be put on the timer, so that some condition needs to be met before the timer will execute the action.

The timer will also signal the state machine that the timer has triggered, so that the state machine can check it's states.

2.2.3.4 Calendar Service

A device can also be configured to execute an action on a specific moment in time, using the calendar service. A calendar task can also be set to repeat by a interval. A filter value for which weekdays the calendar task should execute on is also supported.

The calendar will also signal the state machine that the calendar task has triggered, so that the state machine can check it's states.

The calendar service will only execute when the device has been synchronized with ntp.

2.2.3.5 Action Service

An action can get, set a value on the device itself or on a other device. It can also send a report with the status of one its own values. An action can also start and stop a timer. When an action is set to be executed, it is added to a queue. If the previous action is a similar action to the same partner, then the action is combined with the previous action instead of being added. See Figure 2.3 for the flow chart of the action enqueue process.

2.2.3.6 Calculation Service

A calculation consistes of two sides, left and right, and a operator. A side can be a constant value, a reference to a value, local or from a partner, or an other calcualtion. It can also check if a timer or calculation has executed or if a state machine is in a spefic state. Calculations can be consists of other calculations to generate complex calculations.

2.2.3.7 State Machine Service

A state machine consists of states and transactions. A state has an ID so that it can be referenced. A transaction can have a calculation, action and the state the state machine should go to. If there is no calculation present on the transaction, then it is interpreted as always true. If there is no next state the transaction should take, then the state machine will just keep it's current state.

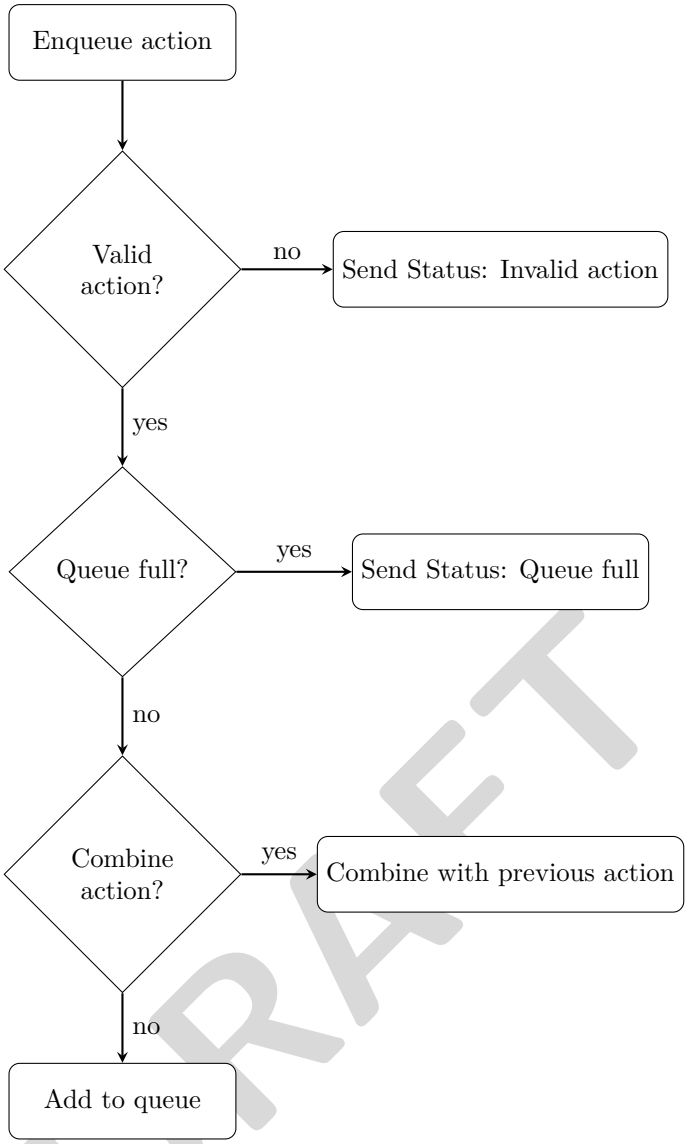

Figure 2.3: Action enqueue

The state machine can be triggered by 4 different events, timer event, calendar event, incoming value report and a local value update. When the state machine is triggered, it will loop though all the state machines and check the current state of each machine. It will execute the first transaction that are true from the current state of the machine. It will only execute each transaction once per event. In this way there will not be any run away loops due to invalid configuration of the state machine. When the statemachine is done handling the event, it will check if there was any state change in the state machines and then evaluate the calculations again, so that any calculations relaying on state machine states can be checked. It will then run though all the state machine again. See Figure 2.4 for the flow chart of state machine execution.

2.2.3.8 Configuration Service

The configuration service is used to persistent and enable the current configuration. When ever the device receives any new configuration, the configuration status is marked as started, and the device sends a Status message with the information about that the status has changed to started. When the configuration is started the state machine, timer and calendar is halted, until the configuration is switch backed to idle again. This can be done either by an incoming configuration mode setting the mode, or by a timeout that happens when there has not been any new configuration for sometime. If the timeout happen the device will do a rollback of any received configurations.

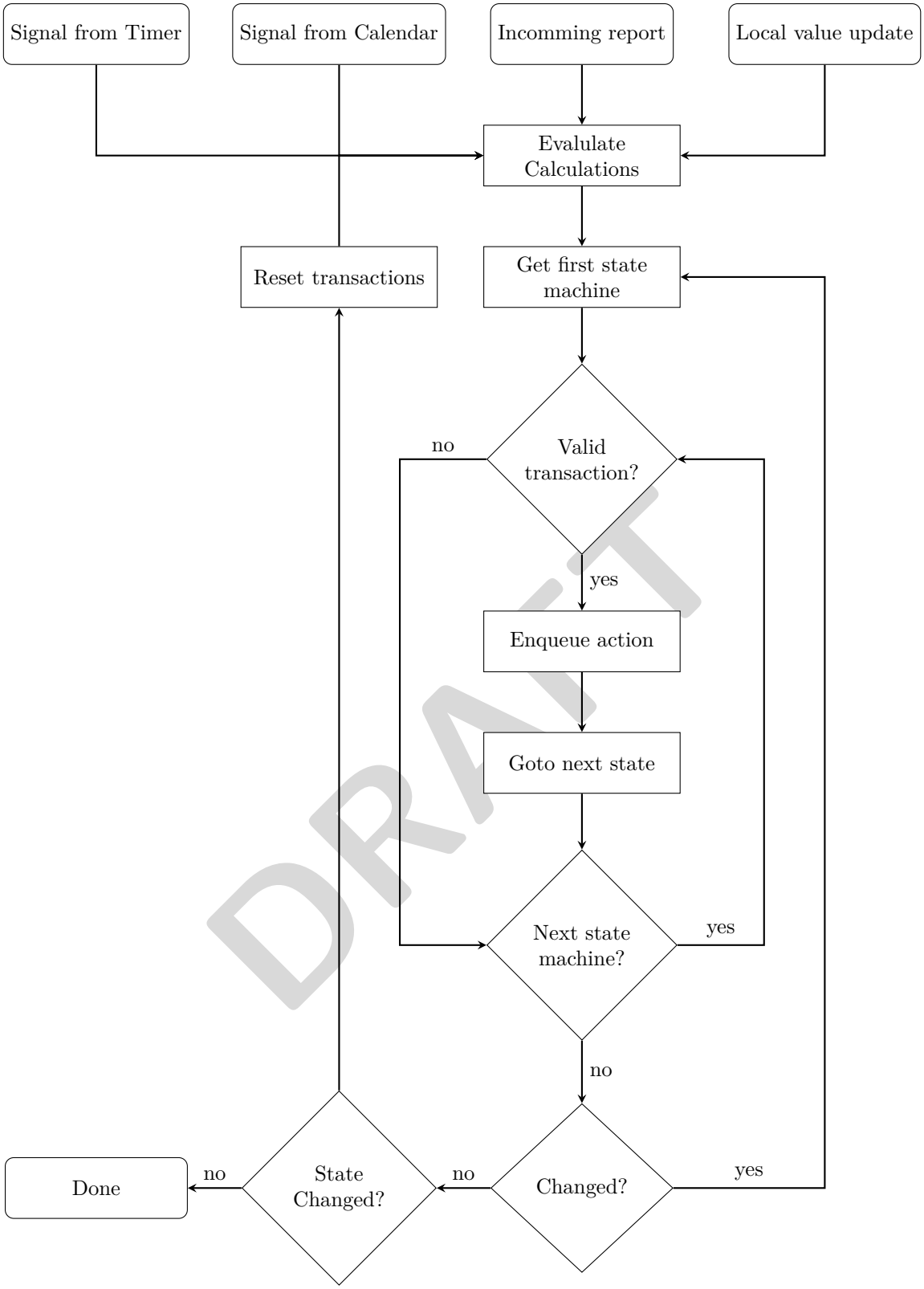

Figure 2.4: State machine executing

2.2.3.9 Status Service

The status service is used to report errors in the device. A status report contains the type of error and a code that describes what happen. There is also an optional data field that can be used to report additional information about the error.

The application can also send error messages using the status service. The application has it own type id, but the code are application specific.

2.2.3.10 Firmware Update Service

The firmware update service is used to updated the bootloader, stack and application of a device. Application can be updated without bootloader and stack, so that changes in application code, can easily be updated on a device, because application code is limited in size. The bootloader can only be updated together with stack and application, because there might be a change in the bootloader so that the stack is placed different in memory or something similar.

2.3 XML description

In this chapter the xml messages will be described. In every section there are descriptions of xml tags and examples of how to use them.

2.3.1 Network and device

The network and device tags are used to encapsulate all the messages. They both have an attribute labeled version. The version on the network tag, describes the version of the network and device tag. The version on the device tag describes which version is used of the message format.

The device tag also has a device ID for addressing multiple devices in a single device and go to sleep for defining the amount of time, before the device should go to sleep.

2.3.1.1 Tag description

The message tag for the network and device is described in Table 2.6.

XML tag	Attribute	Required	Description
`network`	`-`	Yes	Root tag of each message
	`version`	Yes	Version of the network
`device`	`-`	Yes	Current device
	`version`	Yes	Version of the device message
	`device_id`	No	ID of the device If no ID present then the message is for the main device
	`goto_sleep`	No	Time to wait before device goes to sleep

Table 2.6: Network and device tags

2.3.1.2 Example

The following section contains an XML example for using the *network* and *device* tags.

2.3.1.3 Network and device tags

See XML 2.3 for an example of how to encapsulate the messages.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network version="1">
  <device version="1" device_id="1" goto_sleep="10000">
    <.../>
  </device>
</network>
```

XML 2.3: Encapsulation using network and device tags

2.3.2 Network management

The network management message is used to transmit the Network Controller key and the Network key, that are used for encrypting communication between the network controller and a specific device.

The network management message only consists of the *network_include* tag.

2.3.2.1 Tag description

The message tag for the network management is described in Table 2.7

XML tag	Attribute	Required	Description
`network_include`	-	Yes	Is used by the network controller to include a device into the network.

Table 2.7: Network management tags

2.3.2.2 Inclusion data

The inclusion data is RSA encrypted with the public key of the device. The decrypted data consistes of a controller key, the network key AES encrypted with the controller key, and a CRC of the complete message. The format is described in Table 2.4.

Type	Data
Controller key (hex)	`0102030405060708090A0B0C0D0E0F00`
Network key (hex)	`00112233445566778899AABBCCDDEEFF`
Encrypted Network key (hex)	`E9855F797E25CDF8EC9A4ECB83E6632B`
CRC (hex)	`5948`
Complete message (hex)	`0102030405060708090A0B0C0D0E0F00E9855F797E25CDF8EC9A4ECB83E6632B5948`
RSA public key (decimal)	`65537`

RSA common key (decimal)	`5447071546242650082931150031052527271190401635629791278140655528`
	`7424107061733513909763877900966260067194263263804785187364816327`
	`2789081193108015117548459`

RSA encrypted message (hex)	`014CAE536800D9114DB19EEEC7EF857658BBA06F145B00B09ACE371F003BB9C2` `FBF2C500768DEEB22FCAC92DEC605CB998C244EBAE48AD8B10CEE139209796F1`

Table 2.8: The values used in the example

2.3.2.3 Examples

The following section contains a XML example for using the *network_include* messages.

Network include

To include a device a network_include message with a controller key andte network key must be send form the network controller to the device. See XML 2.4 for an example of the include message and see Table 2.8 for the used keys.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="networkmanagement.xsd">
  <device version="1">
    <network_include>
      014CAE536800D9114DB19EEEC7EF857658BBA06F145B00B09ACE371F003BB9C2
      FBF2C500768DEEB22FCAC92DEC605CB998C244EBAE48AD8B10CEE139209796F1
    </network_include>
  </device>
</network>
```

XML 2.4: Network controller key

2.3.3 Public key

The public key message is used to get and report the devices public key.

2.3.3.1 Tag description

The message tag for the public key is described in Table 2.9.

XML tag	Attribute	Required	Description
`publickey_get`	-	Yes	Get the public key from the device
`publickey_report`	-	Yes	Report the public key
	`key_type`	No	The type of the public key. See Table 2.10 for allowed values.

Table 2.9: Public key tags

2.3.3.2 Key types

The supported public key types is listed in Table 2.10.

Key Type Id	Key Type
1	RSA
...	...

Table 2.10: Public key types

2.3.3.3 Examples

The following section contains XML examples for using the *publickey_get* and *publickey_report* messages.

Get a public key

In order to get a public key, a *publickey_get* message should be sent to the device. See XML 2.5 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="public_key.xsd">
  <device version="1">
    <publickey_get/>
  </device>
</network>
```

XML 2.5: Get a public key

Report a public key

In order to report a public key, a *publickey_report* message should be sent from the device. See XML 2.6 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="public_key.xsd">
  <device version="1">
    <publickey_report>45282897238592983093840927 35</publickey_report>
  </device>
</network>
```

XML 2.6: Report a public key

2.3.4 Service description

The service description message is used to obtain and report descriptions of specific services.

The standalone tag *service* for each description allows the user to get a list of supported services for a specific device. A service is described by a *type* tag, which indicates whether the service allows e.g. memory information, device description or value description. Furthermore a *version* tag indicates the highest supported version of this service.

2.3.4.1 Tag description

The message tag for the service description is described in Table 2.11.

XML tag	Required	Description
`service_description_get`	Yes	Get the service description
`service_description_report`	Yes	Report the service description

Table 2.11: Service description tags

The child tags for the *service_description_report* message are described in Table 2.12.

XML tag	Attribute	Required	Description
`service`	`-`	Yes*	The services the device supports *Only required under service_description_report
	`service_id`	Yes	The service ID. See Table 2.13
	`version`	Yes	The maximum supported version of the service

Table 2.12: Service description message child tags

2.3.4.2 Service

The valid options for the *service_id* attributes in the service are listed in Table 2.13.

Service Id	Device Type
1	Public key
2	Memory description
3	Device description
4	Value description
5	Value
6	Partner information
7	Action
8	Calculation
9	Timer
10	Calendar
11	State machine
12	Firmware update
13	Channel Scan
14	Status
15	Configuration
...	...

Table 2.13: Service Description Types

2.3.4.3 Example

The following section contains XML examples for using the *service_description_get* and *service_description_report* messages.

Get a service description

In order to get a service description, a *service_description_get* message should be sent to the device.

See XML 2.7 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="service_description.xsd">
  <device version="1">
    <service_description_get/>
  </device>
</network>
```

XML 2.7: Get a service description

Report a service description

To report a service description, the *service_description_report* message should be sent from the device with its corresponding child tags as described earlier.

See XML 2.8 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="service_description.xsd">
  <device version="1">
    <service_description_report>
      <service service_id="1" version="1"/>
      <service service_id="3" version="1"/>
      <service service_id="4" version="1"/>
      <service service_id="2" version="1"/>
      <service service_id="6" version="1"/>
      <service service_id="5" version="1"/>
      <service service_id="13" version="1"/>
    </service_description_report>
  </device>
</network>
```

XML 2.8: Report a service description

2.3.5 Memory information

The memory information message is used to describe each possible number of timers, actions, etc. for a specific device, and is maintained using a report or get message.

2.3.5.1 Tag description

The message tag for the memory description is described in Table 2.14.

XML tag	Attribute	Required	Description
`memory_information_get`	-	Yes	Get memory information of the different services of the device.
`memory_information_report`	-	Yes	Report memory information of the different services of the device.

Table 2.14: Memory description tags

The child tags for the *memory_information_report* message are described in Table 2.15.

XML tag	Attribute	Required	Description
`memory_information`	-	Yes	The memory the device supports
	`memory_id`	Yes	The service ID, See Table 2.16
	`count`	Yes	The maximum number of the service the device supports
	`free_count`	Yes	The number of free memory slots

Table 2.15: Memory_information_report child tags

2.3.5.2 Memory

The valid options for the *memory_id* attributes in the *memory_information* are listed in Table 2.16.

Memory Id	The count type
1	Value
2	Partner Information
3	Action Items
4	Calculation
5	Timer
6	Calendar
7	Statemachine
8	Statemachine Transactions
...	...

Table 2.16: Memory Types

2.3.5.3 Examples

The following section contains XML examples for using the *memory_information* messages.

Get the memory information

In order to get the memory information, a *memory_information_get* message has to be sent to the device. See XML 2.9 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="memory_information.xsd">
  <device version="1">
    <memory_information_get/>
  </device>
</network>
```

XML 2.9: Get the memory information

Report the memory information

To report the memory information, the *memory_information_report* message has to be sent from the device with its corresponding child tag as described earlier. See XML 2.10 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="memory_information.xsd">
  <device version="1">
    <memory_information_report>
      <memory_information memory_id="1" count="5" free_count="3"/>
      <memory_information memory_id="2" count="10" free_count="8"/>
      <memory_information memory_id="3" count="5" free_count="1"/>
      <memory_information memory_id="4" count="6" free_count="4"/>
      <memory_information memory_id="5" count="8" free_count="6"/>
      <memory_information memory_id="6" count="8" free_count="5"/>
      <memory_information memory_id="7" count="9" free_count="0"/>
      <memory_information memory_id="8" count="3" free_count="1"/>
    </memory_information_report>
  </device>
</network>
```

XML 2.10: Report the memory information

FiXme Note: Add example for service get memory

2.3.6 Device description

The device description message is used to describe information about the device, e.g. type, manufacture, etc. and is maintained using a report, get and set message.

2.3.6.1 Tag description

The message tag for the device description is described in Table 2.17.

XML tag	Attribute	Required	Description
`device_description_get`	-	Yes	This tag is used to request a device description from a device
`device_description_report`	-	Yes	This tag is used to report a device description from a device
`device_description_set`	-	Yes	This tag is used to set some device properties on a device

Table 2.17: Device Description tags

The child tags for the *device_description_report* and *device_description_set* message are described in Table 2.18.

XML tag	Attribute	Required	Description
`info`	-	Yes	Set a device property
	`type_id`	Yes	The id of the info. See Table 2.19
	`number`	No	A number value
	`string`	No	A string value
	`hex`	No	A hex binary value

Table 2.18: Device Description child tags

Id	Name	Type	Description
1	Type	Number	Manufacture type id
2	Manufacturer	Number	The manufacture of the device. See Table 2.22 for a list of manufactures
3	SGTIN	Hex	Serialized Global Trade Identification, which is a unique device number
4	Mac Address	Hex	The mac address of the device, which is a unique device address
5	Hardware Version	String	The version of the hardware
6	Bootloader Version	String	The version of the bootloader
7	Stack Version	String	The version of the stack
8	Application Version	String	The version of the application
9	Protocol	Number	The communication protocol the device uses. See Table 2.20 for a list of protocols
10	Product	Number	Manufacture product id
11	**Included**	Number	Boolean value that indicates if the device is included
12	**Name**	String	The name of the device
13	Radio Mode	Number	Device communication mode. See Table 2.21
14	**Wakeup Interval**	Number	The time in milliseconds that the device is offline between 2 consecutive RX active periods
15	**Wakeup Offset**	Number	The offset in milliseconds needed for the calculation of the RX on period
16	**Wakeup Channel**	Number	The radio channel the device listens on for wakeup frames
17	**Channel Map**	Hex	The current channel map. The channel map will always have 4 channels defined and 2 of them will be the synchrozations channels.
18	Channel Scan Time	Number	The time is takes to scan a single channel
19	IPv6 Address	Hex	IPv6 Address

Table 2.19: Device Description Types. Types in **bold** are settable

Protocol

The different protocols are described in Table 2.20.

Protocol Id	Protocol
1	Lemonbeat
2	Wi-Fi
3	Ethernet
...	...

Table 2.20: Protocols

Radio Mode

The radio mode described how a device communicates over the air. The default value for this is *Always Online*, so this info can be omitted in a *device_description_report* tag, if the uses the default radio mode. The different radio modes are defined in Table 2.21.

Radio Mode Id	Description
0	Always Online
1	Wake on Radio
2	Wake on Event
3	TX Only
4	RX Only

Table 2.21: Radio Mode

Manufacturer

The different manufactures are defined in Table 2.22.

Manufacture Id	Manufacture
1	RWE
2	Seluxit
...	...

Table 2.22: Manufactures

Sgtin

Every device must have a unique identification number that is assigned to the device under the manufacturing process. The identification number of the device must remain constant throughout the devices entire lifetime. The identification number used is a 96 bits Serialized Global Trade Identification Number (SGTIN-96), which is a standard for making identification numbers.

The SGTIN-96 is specified in the "EPC global Tag Data Standards Version 1.4"[1]. The SGTIN-96 consists of a *Header* and five fields: *Filter Value*, *Partition*, *Company Prefix*, *Item Reference* and *Serial Number*, as shown in Table 2.23.

	Header	Filter value	Partition	Company prefix	Item Reference	Serial Number
SGTIN-96	**8 bit**	**3 bit**	**3 bit**	**20-40 bit**	**24-4 bit**	**38 bit**
	00110000 (Binary)	See description below	See description below	999,999 - 999,999,999,999 (Max decimal range)	9,999,999-9 (Max decimal range)	274,877,906,943 (Max decimal value)

Table 2.23: SGTIN-96

The *Filter Value* is not part of the SGTIN-96 pure identity, but is additional data that is used for fast filtering and pre-selection of basic logistics types. The normative specifications for it are specified by GS-1.

[1] `http://www.gs1.org/gsmp/kc/epcglobal/tds/`

Partition is an indication of where the subsequent *Company Prefix* and *Item Reference* numbers are divided. The SGTIN-96 and GS-1 GTIN structure matches each other in which the *Company Prefix* added to the *Item Reference* number (prefixed by the single Indicator Digit). This gives 13 digits in total; the *Company Prefix* can vary from 6 to 12 digits and the concatenation of single Indicator Digit and *Item Reference* from 7 to 1 digit(s).

The values of *Partition* and the corresponding sizes of the *Company Prefix* and *Item Reference* fields are defined in Table 2.24.

Partition Value(P)	**Company Prefix**		**Indicator Digit and Item Reference**	
	Bits (M)	**Digits (L)**	**Bits (N)**	**Digits**
0	40	12	4	1
1	37	11	7	2
2	34	10	10	3
3	30	9	14	4
4	27	8	17	5
5	24	7	20	6
6	20	6	24	7

Table 2.24: SGTIN-96 Partition Values

Channel map

The channel map is a bit map of the selected channels the device uses for communication. The first bit (LSB) is channel 1 and the last bit (MSB) is channel 32. The channels are defined in Table **??**.

2.3.6.2 Examples

The following section contains XML examples for using the *device_description_get* and *device_description_report* messages.

Get a device description

In order to get a device description, a *device_description_get* message has to be sent to the device. See XML 2.11 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="device_description.xsd">
  <device version="1">
    <device_description_get/>
  </device>
</network>
```

XML 2.11: Get the device description

Report a device description

To report a device description, the *device_description_report* message has to be sent from the device with its corresponding attributes as described earlier. See XML 2.12 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="device_description.xsd">
  <device version="1">
    <device_description_report>
      <!-- Type -->
      <info number="1" type_id="1"/>
      <!-- Manufacturer -->
      <info hex="00112233445566778899AABB" type_id="2"/>
      <!-- Sgtin -->
      <info hex="AABBCCDDEEFF" type_id="3"/>
      <!-- Mac Address -->
      <info number="1" type_id="4"/>
      <!-- Hardware Version -->
      <info number="14" type_id="5"/>
      <!-- Bootloader Version -->
      <info number="1" type_id="6"/>
      <!-- Stack Version -->
      <info hex="00340080" type_id="7"/>
      <!-- Application Version -->
      <info number="500" type_id="8"/>
      <!-- Protocol -->
      <info number="10000" type_id="9"/>
      <!-- Product -->
      <info number="150" type_id="10"/>
      <!-- Included -->
      <info number="2" type_id="11"/>
      <!-- Name -->
      <info number="4" type_id="12"/>
      <!-- Radio Mode -->
      <info number="1" type_id="13"/>
      <!-- Wakeup Interval -->
      <info string="Device name" type_id="14"/>
    </device_description_report>
  </device>
</network>
```

XML 2.12: Report of the device description

Set device description properties

In order to set a device description property, the *device_description_set* message has to be sent to the device with the corresponding attributes as described earlier. See XML 2.13 for an example of the set message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="device_description.xsd">
  <device version="1">
    <device_description_set>
      <!-- Included -->
      <info type_id="11" number="1"/>
      <!-- Wakeup Interval -->
      <info type_id="14" number="10000"/>
      <!-- wakeup_offset -->
      <info type_id="15" number="150"/>
      <!-- wakeup_channel -->
      <info type_id="16" number="2"/>
      <!-- name -->
      <info type_id="12" string="New value name"/>
    </device_description_set>
  </device>
</network>
```

XML 2.13: Set the device description property

2.3.7 Value Description

The value description message is used to describe each possible value for a specific device, e.g. type, mode etc. and is accessed using a report or get message.

A number value has *min*, *max* and *step*. If a value is out side of the min-max range, then it is capped to be with in this range. If the value do not conform the the step size, then the value is rounded down.

2.3.7.1 Tag description

The message tag for the value description is described in Table 2.25.

XML tag	Attribute	Required	Description
`value_description_get`	`-`	Yes	Get the description value
	`value_description_id`	No	ID of the description value To get all the value descriptions omit value_description_id
`value_description_report`	`-`	Yes	Report the description value
`value_description_add`	`-`	Yes	Creates a virtual value
`value_description_delete`	`-`	Yes	Delete a virtual value
	`value_description_id`	No	The id of the value description that should be deleted
`value_description_get_memory`	`-`	Yes	Request a memory report for this service
`value_description_report_memory`	`-`	Yes	Report the memory information for this service
	`count`	Yes	The maximum number of value descriptions the device supports
	`free_count`	Yes	The used number of value descriptions

Table 2.25: Value Description tags

The child tags for the *value_description_report* and *value_description_add* message are described in Table 2.26.

XML tag	Attribute	Required	Description
`value_description`	`-`	Yes	The value description
	`value_id`	Yes	ID of the description value
	`type_id`	Yes	Specifies type of value, see Table 2.29
	`mode`	Yes	Possible interaction with value: 1. Read Only (Only read from the device) 2. Read/Write (Read and write to/from the device) 3. Write Only (Write to the device)
	`persistent`	Yes	Specifies if the value is persistent doing a power cycling 0. Not persistent 1. Is persistent
	`name`	No	The name of the device
	`min_log_interval`	No	Minimum time between log values in seconds
	`max_log_values`	No	Maximum numbers of stored values for this device
	`virtual`	No	Specifies if the value is a virtual value

Table 2.26: Value_description_report message child tags

The child tags for the *value_description* in Table 2.27. The sections that follow will describe all child tags in more detail.

XML tag	Attribute	Required	Description
`number_format`	`-`	Yes*	When the format of the value is a number *If not defined, then another format tag must be defined
	`unit`	Yes	Specifies which unit the value holds
	`min`	Yes	Minimum value
	`max`	Yes	Maximum value
	`step`	Yes	The step-size by which the value can
`string_format`	`-`	Yes*	When the format of the value is a string *If not defined, then another format tag must be defined
	`max_length`	Yes	Maximum length of the string
`hexBinary_format`	`-`	Yes*	When the format of the value is hex binary *If not defined, then another format tag must be defined
	`max_length`	Yes	The maximum length of the hex binary data

Table 2.27: Value description child tags

The child tags for the string_format are described in Table 2.28.

XML tag	Required	Description
valid_value	No	The values that are valid for the string value

Table 2.28: Value description String format child tags

Type

The valid options for the *type_id* attribute in the value description are listed in Table 2.29. The *type_id* attribute is a required and should be set to a valid option.

Unit

The *unit* attribute is in the International System of Units (SI) format. The *unit* attribute is a required and must be set to a valid value.

Type Id	Name
1	Temperature
2	Luminance
3	Power
4	Electricity
5	Humidity
6	Velocity
7	Direction
8	Atmospheric
9	Barometric
10	Solar Radiation
11	Dew Point
12	Rain Rate
13	Tide level
14	ON/OFF
15	Awake State
16	Event
17	General Purpose
18	Counter
19	Energy
20	Level
21	CO2
22	Air Flow
23	Tank Capacity
24	Distance
25	Climate Control
26	Program
27	Fan Speed
28	Error Code
29	Operation Mode
30	Louvre
31	Mode
32	Time
33	Duty Cycle
34	Voltage
35	Current
36	Frequency
37	Battery
38	Timezone Offset
39	Year
40	Month
41	Day Of Month
42	Weekday
43	Hour
44	Minute
...	...

Table 2.29: Value description types

2.3.7.2 Examples

The following section contains XML examples for using the *value_description_get*, *value_description_report*, *value_description_a* and *value_description_delete* messages.

Get a specific value description

In order to get a specific value description, a *value_description_get* message has to be sent to the device with a value description ID. See XML 2.14 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_get value_description_id="1"/>
  </device>
</network>
```

XML 2.14: Get the value description with value description ID 1

Get all value descriptions

In order to get all value description, a *value_description_get* message has to be sent to the device without a value description ID. See XML 2.15 for an example of the get all message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_get/>
  </device>
</network>
```

XML 2.15: Get all value descriptions

Report a specific value description for number format

To report a specific value description for number format, the *value_description_report* message has to be sent from the device with its child tag *value_description* and its child tag *number_format*. See XML 2.16 for an example of the report for number format message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_report>
      <value_description value_id="1" type_id="14" mode="2" persistent="0" name="light"
    min_log_interval="300" max_log_values="20">
        <number_format unit="W" min="0" max="1" step="1"/>
      </value_description>
    </value_description_report>
  </device>
</network>
```

XML 2.16: Report of the value description for number format

Report a specific value description for string format

To report a specific value description for string format, the *value_description_report* message has to be sent from the device with its child tag "value_description" and its child tag *string_format*. See XML 2.17 for an example of the report string format message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_report>
      <value_description value_id="2" type_id="26" mode="1" persistent="1" name="washing program">
        <string_format max_length="20">
          <valid_value>wool</valid_value>
          <valid_value>cotton</valid_value>
        </string_format>
      </value_description>
    </value_description_report>
  </device>
</network>
```

XML 2.17: Report of the value description for string format

Add a virtual value description

To add a virtual value description to a device, the *value_description_add* message has to be sent to the device. See XML 2.18 for an example of the add message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_add>
      <value_description value_id="5" type_id="14" mode="2" persistent="0" name="Virtual Value"
    virtual="1">
        <number_format unit="W" min="0" max="100" step="1"/>
      </value_description>
    </value_description_add>
  </device>
</network>
```

XML 2.18: Add virtual value description

Delete a virtual value description

To delete a virtual value description from a device, the *value_description_delete* message has to be sent to the device. See XML 2.20 for an example of the delete message.

«««< local

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_delete value_description_id="5" />
  </device>
</network>
```

XML 2.19: Delete virtual value description

======= »»»> other

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value_description.xsd">
  <device version="1">
    <value_description_delete value_description_id="5" />
  </device>
</network>
```

XML 2.20: Delete virtual value description

2.3.8 Value

The value message is used to get, set, report and logging a value and is maintained using a report, get, set and get log message.

If the device is not synchronized with NTP, then the reported timestamp will be zero.

2.3.8.1 Tag description

The message tag for the value is described in Table 2.30.

XML tag	Attribute	Required	Description
value_get	-	Yes	The value_get tag is used to get all the values. See examples below.
	value_id	No	ID of the value. To get all the values omit value_id
value_report	-	Yes	The value_report tag is used to report all the values. See examples below.
	value_id	Yes	ID of the value
	number	Yes*	The number value *If not defined string or hex must be defined
	string	Yes*	The string value *If not defined number or hex must be defined
	hex	Yes*	The hex value *If not defined number or string must be defined
	timestamp	Yes	Timestamp of when the current value was send.
value_set	-	Yes	The value_set tag is used to set multiple the values. See examples below.
	value_id	Yes	ID of the value
	number	Yes*	The number value *If not defined string or hex must be defined
	string	Yes*	The string value *If not defined number or hex must be defined
	hex	Yes*	The hex value *If not defined number or string must be defined
	timestamp	Yes	Timestamp of when the current value was send.
value_get_log	-	Yes	The value_get_log tag is used to get all the logged values. See examples below.
	Value_id	No	ID of the value To request all the logged values omit value_id
	start_time	No	To request the logged values with a timestamp larger or equal to this value. The start_time has the same type as the timestamp in the value_report tag.
	log_count	No	To request up to log count number of logged events.

Table 2.30: Value tags

2.3.8.2 Examples

This flowering section consists of examples in xml for how to use the "value_get", "value_report", "value_report", "value_set", "value_set" and "value_get_log" messages.

Get a specific value

In order to get a specific value, a "value_get" message has to be sent to the device with a value ID. See XML 2.21 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_get value_id="1"/>
  </device>
</network>
```

XML 2.21: Get the value with ID 1

Get all values

In order to get all values, a "value_get" message has to be sent to the device without a value ID. See XML 2.22 for an example of the "get all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_get/>
  </device>
</network>
```

XML 2.22: Get all values

Report a specific number value

To report a specific number value, the "value_report" message has to be sent from the the device with a value ID and a number. See XML 2.23 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_report value_id="1" timestamp="0" number="55"/>
  </device>
</network>
```

XML 2.23: Report of the value with ID 1 has the value 55

Report a specific string value

To report a specific string value, the "value_report" message has to be sent from the the device with a value ID and a string. See XML 2.24 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_report value_id="2" string="ON" timestamp="0"/>
  </device>
</network>
```

XML 2.24: Report of the value with ID 2 has the value ON

Report all values

To report all values, the "value_report" message has to be sent from the device with a value ID and a number or a string. See XML 2.25 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_report value_id="1" number="55" timestamp="0"/>
    <value_report value_id="2" string="ON" timestamp="0"/>
  </device>
</network>
```

XML 2.25: Report of all the values

Set a specific number value

In order to set a specific number value, a "value_set" message has to be sent with a value ID and a number. See XML 2.26 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_set value_id="1" timestamp="0" number="55"/>
  </device>
</network>
```

XML 2.26: Set the value with ID 1 to 55

Set a specific string value

In order to set a specific string value, a "value_set" message has to be sent with a value ID and a string. See XML 2.27 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_set value_id="2" timestamp="0" string="ON"/>
  </device>
</network>
```

XML 2.27: Set the value with ID 2 to ON

Set multiple values

In order to set multiple values, a "value_set" with a value ID and a number or a string. See XML 2.28 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_set value_id="1" timestamp="0" number="55"/>
    <value_set value_id="2" timestamp="0" string="ON"/>
  </device>
</network>
```

XML 2.28: Set the value with ID 1 to 55 and the value with ID 2 to ON

Get Log values for a specific value

In order to get a log for a specific value, a "value_get_log" message has to be sent to the device with a value ID, a start time and a log count. See XML 2.29 for an example of the "get log" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_get_log value_id="1" start_time="1314980580" log_count="2"/>
  </device>
</network>
```

XML 2.29: Get the log for the value with ID 1

Report log values for a specific value

To report all logs for a specific value, the "value_report" message has to be sent from the device with a value ID, a timestamp and a number or a string See XML 2.30 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_report value_id="1" timestamp="1314980580" number="55"/>
    <value_report value_id="2" timestamp="1314980585" string="ON"/>
  </device>
</network>
```

XML 2.30: Report all the log values for the value with ID 1 and 2

2.3.9 Partner information

The partner information message is used to obtain detailed information about existing partners and groups as well as creating additional information or deleting information from these.

The partner information message is maintained by get, set, report and delete tags. Each partner information is labeled with a "partner id" for identification purposes, and contains either a partner or group.

A Partner is described by an address, encryption key and radio mode, where as a group is described as list of partners for the purpose of sending message to all partners in the group at once, through a multicast address.

2.3.9.1 Tag description

The message tag for the partner information is described in Table 2.31.

XML tag	Attribute	Required	Description
`partner_information_get`	-	Yes	The partner_information_get tag is used to get partners. See examples below.
	`partner_id`	No	ID of the partner. To get all the partners omit partner_id
`partner_information_report`	-	Yes	The partner_information_report tag is used to report partners. See examples below.
`partner_information_set`	-	Yes	The partner_information_set tag is used to set partners. See examples below.
`partner_information_delete`	-	Yes	The partner_information_delete tag is used to delete partners. See examples below.
	`partner_id`	No	ID of the partner. To delete all the partners omit partner_id.
`partner_information_get_memory`	-	Yes	This tag is used to request the partner information memory information from a device
`partner_information_report_memory`	-	Yes	This tag is used to report the partner information memory information from a device
	`count`	Yes	The maximum number of partners the device supports
	`free_count`	Yes	The used number of partners

Table 2.31: Partner tags

The child tags for the partner_information_report and partner_information_set message are described in Table 2.32.

XML tag	Attribute	Required	Description
`partner`	-	Yes*	Describes a partners address, encryption key and radio mode. See examples below. *If not defined group must be defined
	`partner_id`	Yes	ID of the partner
`group`	-	Yes*	Is used to group partners to a multicast address *If not defined partner must be defined
	`partner_id`	Yes	ID of the partner

Table 2.32: Partner information child tags

The child tags for partner are described in Table 2.33.

XML tag	Attribute	Required	Description
`info`	`-`	Yes	Set a device property
	`type_id`	Yes	The id of the info. See Table 2.34
	`number`	No	A number value
	`string`	No	A string value
	`hex`	No	A hex binary value

Table 2.33: Partner Information partner child tags

The valid values for *type_id* in partner information is desribed in Table 2.34.

Id	Name	Type	Description
13	Radio Mode	Number	The partners radio mode. See Table 2.21
14	Wakeup Interval	Number	The offset in milliseconds of when the partner starts to wakeup
15	Wakeup Offset	Number	How often in milliseconds the partner will wakeup
16	Wakeup Channel	Number	The radio channel the partner listens on to see if it has to wakeup
17	Channel Map	Hex	The current channel map
18	Channel Scan Time	Number	The time is takes to scan a single channel
19	IPv6 Address	Hex	The full IPv6 address of the partner
20	Wakeup Now	Number	The time in milliseconds the partner should be a wake

Table 2.34: Device Description Types. Types in **bold** are settable

The child tags for the group are described in Table 2.35.

XML tag	Attribute	Required	Description
`partner`	`-`	Yes	The partner in the group
	`partner_id`	Yes	ID of the partner

Table 2.35: Group child tags

2.3.9.2 Examples

The following section contains XML examples for using the "partner_information_get", "partner_information_report", "partner_information_set" and "partner_information_delete" messages.

Get a specific partner

In order to get a specific partner, a "partner_information_get" message has to be sent to the device with a partner ID. See XML 2.31 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_get partner_id="1"/>
  </device>
</network>
```

XML 2.31: Get the partner with the ID 1

Get all partners

In order to get all partners, a "partner_information_get" message has to be sent to the device without a partner ID. See XML 2.32 for an example of the "get all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_get/>
  </device>
</network>
```

XML 2.32: Get all partners

Report a specific partner

To report a specific partner, the "partner_information_report" message has to be sent from the device with a partner ID on its corresponding child tags as described earlier. See XML 2.33 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_report>
      <partner partner_id="1">
        <!-- Radio Mode -->
        <info type_id="13" number="1"/>
        <!-- Wakeup Interval -->
        <info type_id="14" number="10000"/>
        <!-- wakeup_offset -->
        <info type_id="15" number="150"/>
        <!-- wakeup_channel -->
        <info type_id="16" number="2"/>
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc00000000000000010000000000000012"/>
      </partner>
    </partner_information_report>
  </device>
</network>
```

XML 2.33: Report for the partner with ID 1

Report all partners

To report all partners, the "partner_information_report" message has to be sent from the device with a partner ID and its corresponding child tags as described earlier. See XML 2.34 for an example of the "report all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_report>
      <partner partner_id="1">
        <!-- Radio Mode -->
        <info type_id="13" number="1"/>
        <!-- Wakeup Interval -->
        <info type_id="14" number="10000"/>
        <!-- wakeup_offset -->
        <info type_id="15" number="150"/>
        <!-- wakeup_channel -->
        <info type_id="16" number="2"/>
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc00000000000000100000000000012"/>
      </partner>
      <partner partner_id="2">
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc00000000000000100000000000017"/>
      </partner>
      <group partner_id="3">
        <partner partner_id="1"/>
        <partner partner_id="2"/>
      </group>
    </partner_information_report>
  </device>
</network>
```

XML 2.34: Report all partners

Set a specific partner

In order to set a specific partner, a "partner_information_set" message has to be sent to the device with a partner ID on its corresponding child tags as described earlier. See XML 2.35 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_set>
      <partner partner_id="1">
        <!-- Radio Mode -->
        <info type_id="13" number="1" />
        <!-- Wakeup Interval -->
        <info type_id="14" number="10000"/>
        <!-- wakeup_offset -->
        <info type_id="15" number="150"/>
        <!-- wakeup_channel -->
        <info type_id="16" number="2"/>
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc00000000000000100000000000012"/>
      </partner>
    </partner_information_set>
  </device>
</network>
```

XML 2.35: Set the partner with ID 1

Set multiple partners

In order to set multiple partners, a "partner_information_set" message has to be sent to the device with a partner and its corresponding child tags as described earlier. See XML 2.36 for an example of the "set multiple" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_set>
      <partner partner_id="1">
        <!-- Radio Mode -->
        <info type_id="13" number="1" />
        <!-- Wakeup Interval -->
        <info type_id="14" number="10000"/>
        <!-- wakeup_offset -->
        <info type_id="15" number="150"/>
        <!-- wakeup_channel -->
        <info type_id="16" number="2"/>
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc00000000000000100000000000012"/>
      </partner>
      <partner partner_id="2">
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc00000000000000100000000000017"/>
      </partner>
      <group partner_id="3">
        <partner partner_id="1"/>
        <partner partner_id="2"/>
      </group>
    </partner_information_set>
  </device>
</network>
```

XML 2.36: Set multiple partners

Delete a specific partner

In order to delete a specific partner, a "partner_information_delete" message has to be sent to the device with a partner ID. See XML 2.37 for an example of the "delete" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_delete partner_id="1"/>
  </device>
</network>
```

XML 2.37: Delete the partners with ID 1

Delete all partners

In order to delete all partners, a "partner_information_delete" message has to be sent to the device without a partner ID. See XML 2.38 for an example of the "delete all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="partner_information.xsd">
  <device version="1">
    <partner_information_delete/>
  </device>
</network>
```

XML 2.38: Delete all partners

2.3.10 Action

The action message is used to obtain a list of all current actions on a specific device.

The action message is maintained by get, report, set and delete tags. Each action specifies a value which can be both get and set by their corresponding child tags. Furthermore it is possible to group different actions together, as well as setting utilizing a timer to determine the run-time of a specific action.

2.3.10.1 Tag description

The message tag for the action is described in Table 2.36.

XML tag	Attribute	Required	Description
`action_get`	-	Yes	The action_get tag is used to get actions See examples below.
	`action_id`	No	ID of the action To get all the actions omit action_id
`action_report`	-	Yes	The action_report tag is used to report actions. See examples below.
`action_set`	-	Yes	The action_set tag is used to set actions. See examples below.
`action_delete`	-	Yes	The action_delete tag is used to delete actions. See examples below.
	`action_id`	No	ID of the action To delete all the actions omit action_id
`action_invoke`	-	Yes	This tag is used to execute an action directly
	`action_id`	Yes	ID of the action to execute
`action_get_memory`	-	Yes	This tag is used to request the action memory information from a device
`action_report_memory`	-	Yes	This tag is used to report the action memory information from a device
	`count`	Yes	The maximum number of actions the device supports
	`free_count`	Yes	The used number of actions

Table 2.36: Action tags

The child tags for the action_report and action_set message are described in Table 2.37.

XML tag	Attribute	Required	Description
`action`	-	Yes	The action
	`action_id`	Yes	ID of the action

Table 2.37: Action_report and action_set child tags

The child tags for the action are described in Table 2.38.

XML tag	Attribute	Required	Description
`get`	`-`	No	Get the value for the specified value on the specified device
	`value_id`	Yes	ID of the value
	`partner_id`	No	ID for the device where the value should be retrieved from. If no partner_id is present then the value is from the device itself.
	`transport_mode`	No	The transport mode the action should be send with. See Table 2.39
`set`	`-`	No	Set the value of the specified device.
	`value_id`	Yes	ID of the value
	`partner_id`	No	ID for the device where the value should be set. If no partner_id is present then the value will be set on the device itself.
	`number`	Yes*	The number value *If not defined string, number or calculation_id must be defined
	`string`	Yes*	The string value *If not defined number, hexBinary or calculation_id must be defined
	`hexBinary`	Yes*	The hex value *If not defined number, string or calculation_id must be defined
	`calculation_id`	Yes*	The calculation ID *If not defined number or string must be defined
	`transport_mode`	No	The transport mode the action should be send with. See Table 2.39
`report`	`-`	No	Sending a report to the specified device.
	`my_value_id`	Yes	ID of the value from the device itself that is used in the report
	`partner_id`	No	ID for the device where the report should be sent to. If no partner_id is present then the report will be sent to the device itself.
	`transport_mode`	No	The transport mode the action should be send with. See Table 2.39
`timer_start`	`-`	Yes	Start a timer
	`timer_id`	Yes	ID of the timer
`timer_stop`	`-`	Yes	Stop a timer
	`timer_id`	Yes	ID of the timer

Table 2.38: Action child tags

Transport mode

Mode	Name	Description
0	UDP	Fast and simple transport mode
1	TCP	Slow but more reliable transport mode

Table 2.39: Transport mode

2.3.10.2 Examples

The following section contains XML examples for using the "action_get", "action_report", "action_set" and "action_delete" messages.

Get a specific action

In order to get a specific action, an "action_get" message has to be sent to the device with an action ID. See XML 2.39 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_get action_id="1"/>
  </device>
</network>
```

XML 2.39: Get the action with ID 1

Get all actions

In order to get all actions, an "action_get" message has to be sent to the device without an action ID. See XML 2.40 for an example of the "get all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_get/>
  </device>
</network>
```

XML 2.40: Get all actions

Report an action

To report a specific action, the "action_report" message has to be sent from the device with an action ID and its corresponding child tags as described earlier. See XML 2.41 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="2">
        <set value_id="3" partner_id="23" number="12"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.41: Report the set action with ID 2

Report all actions

To report all actions, the "action_report" message has to be sent from the device with an action ID and its corresponding child tags as described earlier. See XML 2.42 for an example of the "report all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <get value_id="2" partner_id="34"/>
        <set value_id="2" partner_id="34" number="12"/>
      </action>
      <action action_id="3">
        <report my_value_id="4" partner_id="12"/>
      </action>
      <action action_id="4">
        <timer_start timer_id="1"/>
        <timer_stop timer_id="2"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.42: Report all actions

Set a specific action

In order to set a specific action, an "action_set" message has to be sent to the device with an action ID and its corresponding child tags as described earlier. See XML 2.43 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_set>
      <action action_id="1">
        <get value_id="2" partner_id="34"/>
      </action>
    </action_set>
  </device>
</network>
```

XML 2.43: Set the action with ID 1

Set multiple actions

In order to set multiple actions, an "action_set" message has to be sent to the device with an action ID and its corresponding child tags as described earlier. See XML 2.44 for an example of the "set multiple" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_set>
      <action action_id="1">
        <get value_id="2" partner_id="34"/>
        <set value_id="2" partner_id="34" string="ON"/>
      </action>
      <action action_id="3">
        <set value_id="4" partner_id="32" calculation_id="1"/>
        <report my_value_id="4" partner_id="12"/>
      </action>
      <action action_id="4">
        <timer_start timer_id="1"/>
        <timer_stop timer_id="2"/>
      </action>
    </action_set>
  </device>
</network>
```

XML 2.44: Set multiple actions

Delete a specific action

In order to delete a specific action, an "action_delete" message has to be sent to the device with an action ID. See XML 2.45 for an example of the "delete" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_delete action_id="5"/>
  </device>
</network>
```

XML 2.45: Delete the action with ID 5

Delete all actions

In order to delete all actions, an "action_delete" message has to be sent to the device without an action ID. See XML 2.46 for an example of the "delete all" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_delete/>
  </device>
</network>
```

XML 2.46: Delete all actions

2.3.11 Calculation

The calculation message is used to perform specific calculations.

The calculation message is maintained by get, report, set and delete tags. Each calculation can be the result of an addition, subtraction, multiplication or division, defined by the Method attribute. The result of a calculation is split into two parts, a left and right which can be readout separately.

2.3.11.1 Tag description

The message tag for the calculation is described in Table 2.40.

XML tag	Attribute	Required	Description
`calculation_get`	-	Yes	The calculation_get tag is used to get calculations. See examples below.
	`calculation_id`	No	ID of the calculation. To get all the calculation omit calculation_id
`calculation_report`	-	Yes	The calculation_report tag is used to report calculations. See examples below.
`calculation_set`	-	Yes	The calculation_set tag is used to set calculations. See examples below.
`calculation_delete`	-	Yes	The calculation_delete tag is used to delete calculations. See examples below.
	`calculation_id`	No	ID of the calculation. To delete all the calculation omit calculation_id
`calculation_get_memory`	-	Yes	This tag is used to request the calculation memory information from a device
`calculation_report_memory`	-	Yes	This tag is used to report the calculation memory information from a device
	`count`	Yes	The maximum number of calculations the device supports
	`free_count`	Yes	The used number of calculations

Table 2.40: Calculation tags

The child tags for the calculation_report and calculation_set message are described in Table 2.41.

XML tag	Attribute	Required	Description
`calculation`	-	Yes	The calculation tag is used define a single calculation
	`calculation_id`	Yes	ID of the calculation
	`method_id`	Yes	The calculation method is used to calculate the result using the two values, left and right. See Table 2.42

Table 2.41: Calculation message child tags

The child tags for the calculation are described in Table 2.42.

XML tag	Attribute	Required	Description
`left / right`	`-`	Yes	The left or right value of the calculation
	`value_id`	Yes*	ID of the value *If no value_id is present then calculation_id must be defined
	`calculation_id`	Yes*	ID of the calculation* If no calculation_id is present then value_id must be defined
	`partner_id`	No	ID for the device with the value If no partner_id is present then the value is from the device itself.
	`statemachine_id`	No	ID of the state machine
	`timer_id`	Yes	ID of the timer
	`calendar_id`	Yes	ID of the calendar task
	`constant_string`	No	A constant string value
	`constant_number`	No	A constant number value
	`constant_hexBinary`	No	A constant hex value
	`is_updated`	No	If this is set to the value 1, then the calculation will return true if the value is updated.

Table 2.42: Calculation child tags

Method

The valid options for the "method_id" attribute in the calculation are listed in Table 2.43. The "method_id" attribute is a required and should be set to a valid option.

Method Id	Method
1	Add
2	Subtract
3	Multiply
4	Divide
5	Modulo
6	Equal
7	Not Equal
8	Smaller
9	Greater
10	Smaller or equal
11	Greater or equal
12	And
13	Or
...	...

Table 2.43: Method types

2.3.11.2 Examples

The following section contains XML examples for using the "calculation_get", "calculation_report", "calculation_set" and "calculation_delete" messages.

Get a specific calculation

In order to get a specific calculation, a "calculation_get" message has to be sent to the device with a calculation ID. See XML 2.47 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_get calculation_id="1"/>
  </device>
</network>
```

XML 2.47: Get the calculation with ID 1

Get all calculations

In order to get all calculations, a "calculation_get" message has to be sent to the device without a calculation ID. See XML 2.48 for an example of the "get all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_get/>
  </device>
</network>
```

XML 2.48: Get all calculations

Report a specific calculation

To report a calculation, the "calculation_report" message has to be sent from the device with its corresponding child tags as described earlier. See XML 2.49 for an example of the "report" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_report>
      <calculation calculation_id="2" method_id="3">
        <left calculation_id="1"/>
        <right constant_number="4"/>
      </calculation>
    </calculation_report>
  </device>
</network>
```

XML 2.49: Report the calculation multiply-method

Report all calculations

To report all calculations, the "calculation_report" message has to be sent from the device with its corresponding child tags as described earlier. See XML 2.50 for an example of the "report all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_report>
      <calculation calculation_id="1" method_id="1">
        <left value_id="1"/>
        <right value_id="1" partner_id="1"/>
      </calculation>
      <calculation calculation_id="2" method_id="3">
        <left calculation_id="1"/>
        <right constant_number="4"/>
      </calculation>
      <calculation calculation_id="3" method_id="6">
        <left value_id="1"/>
        <right constant_number="1"/>
      </calculation>
    </calculation_report>
  </device>
</network>
```

XML 2.50: Report all calculations

Set a specific calculation

In order to set a specific calculation, a "calculation_set" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.51 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_set>
      <calculation calculation_id="1" method_id="1">
        <left value_id="1"/>
        <right value_id="1" partner_id="1"/>
      </calculation>
    </calculation_set>
  </device>
</network>
```

XML 2.51: Set the calculation add-method

Set multiple calculations

In order to set multiple calculations, a "calculation_set" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.52 for an example of the "set multiple" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_set>
      <calculation calculation_id="1" method_id="1">
        <left value_id="1"/>
        <right value_id="1" partner_id="1"/>
      </calculation>
      <calculation calculation_id="2" method_id="3">
        <left calculation_id="1"/>
        <right constant_number="4"/>
      </calculation>
      <calculation calculation_id="3" method_id="6">
        <left value_id="1"/>
        <right constant_string="1"/>
      </calculation>
    </calculation_set>
  </device>
</network>
```

XML 2.52: Set multiple calculations

Delete a specific calculation

In order to delete a specific calculation, a "calculation_delete" message has to be sent to the device with a calculation ID. See XML 2.53 for an example of the "delete" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_delete calculation_id="1"/>
  </device>
</network>
```

XML 2.53: Delete the calculation with ID 1

Delete all calculations

In order to delete all calculations, a "calculation_delete" message has to be sent to the device without a calculation ID. See XML 2.54 for an example of the "delete all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_delete/>
  </device>
</network>
```

XML 2.54: Delete all calculations

FiXme Note: Add is_updated example

2.3.12 Timer

The timer message is used to manage a list of conditional timers. The timer message is maintained by get, report, set and delete tags.

Each timer will performed a specified action after the set number of milliseconds defined by the *after* attribute. Furthermore a timer will only execute the action, if a specific calculation is met as defined by the *calculation_id* attribute. Both are attributes of the child tag *Execute*.

2.3.12.1 Tag description

The message tag for the timer is described in Table 2.44.

XML tag	Attribute	Required	Description
`timer_get`	-	Yes	The timer_get tag is used to get timers. See examples below.
	`timer_id`	No	ID of the timer To get all the timers omit timer_id
`timer_report`	-	Yes	The timer_report tag is used to report timers. See examples below.
`timer_set`	-	Yes	The timer_set tag is used to set timers. See examples below.
`timer_delete`	-	Yes	The timer_delete tag is used to delete timers. See examples below.
	`timer_id`	No	ID of the timer To delete all the timers omit timer_id
`timer_get_memory`	-	Yes	This tag is used to request the timer memory information from a device
`timer_report_memory`	-	Yes	This tag is used to report the timer memory information from a device
	`count`	Yes	The maximum number of timers the device supports
	`free_count`	Yes	The used number of timers

Table 2.44: Timer tags

The child tags for the timer_report and timer_set message are described in Table 2.45.

XML tag	Attribute	Required	Description
`execute`	-	Yes	The timer will execute the action after the specified milliseconds in the attribute after
	`timer_id`	Yes	ID of the timer
	`after`	Yes	Number of milliseconds before execution
	`calculation_id`	No	ID of the calculation. The timer will only execute if the calculation is true
	`action_id`	No	ID of the action that will be executed. If the no action_id is present then an event will be sent to the state machine.

Table 2.45: Timer message child tags

2.3.12.2 Examples

The following section contains XML examples for using the "timer_get", "timer_report", "timer_set" and "timer_delete" messages.

Get a specific timer

In order to get a specific timer, a "timer_get" message has to be sent to the device with a timer ID. See XML 2.55 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_get timer_id="1"/>
  </device>
</network>
```

XML 2.55: Get the timer with ID 1

Get all timers

In order to get all timers, a "timer_get" message has to be sent to the device without a timer ID. See XML 2.56 for an example of the "get all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_get/>
  </device>
</network>
```

XML 2.56: Get all the timers

Report timer with an action

To report a timer with an action, the "timer_report" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.57 for an example of the "report with an action" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_report>
      <execute timer_id="1" after="2000" action_id="2"/>
    </timer_report>
  </device>
</network>
```

XML 2.57: Report for the timer with ID 1 that will execute action 2 after 2000 ms

Report timer without an action

To report a timer without an action, the "timer_report" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.58 for an example of the "report without an action" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_report>
      <execute timer_id="2" after="4000"/>
    </timer_report>
  </device>
</network>
```

XML 2.58: Report for the timer with ID 2 that will execute after 4000 ms

Report all timers

To report all timers, the "timer_report" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.59 for an example of the "report all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_report>
      <execute timer_id="1" after="2000" action_id="2"/>
      <execute timer_id="2" after="4000"/>
    </timer_report>
  </device>
</network>
```

XML 2.59: Report all timers

Set a specific timer

In order to set a timer, a "timer_set" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.60 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_set>
      <execute timer_id="2" after="2000" action_id="3"/>
    </timer_set>
  </device>
</network>
```

XML 2.60: Set the timer with ID 2 to execute action 3 after 2000 ms

Set multiple timers

In order to set multiple timers, a "timer_set" message has to be sent to the device with its corresponding child tags as described earlier. See XML 2.61 for an example of the "set multiple" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_set>
      <execute timer_id="1" after="2000" action_id="2"/>
      <execute timer_id="2" after="4000"/>
    </timer_set>
  </device>
</network>
```

XML 2.61: Set the timers with ID 1 and 2

Delete a specific timer

In order to delete a specific timer, a "timer_delete" message has to be sent to the device with a timer ID. See XML 2.62 for an example of the "delete" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_delete timer_id="1"/>
  </device>
</network>
```

XML 2.62: Delete the timer with ID 1

Delete all timers

In order to delete all timers, a "timer_delete" message has to be sent to the device without a timer ID. See XML 2.63 for an example of the "delete all" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_delete/>
  </device>
</network>
```

XML 2.63: Delete all the timers

2.3.13 Calendar

The calendar message is used to obtain a list of scheduled tasks.

The calendar message is maintained by get, report, set and delete tags. Each task in the calendar will execute at the date and time set by the *start* attribute, and continue until the end date is reached which is set by the *end* attribute. It is possible to define that a task in the calendar should be repeated either with specific number seconds between each pass. The *weekdays* attribute is a filter for which weekdays the repeated task should trigger.

2.3.13.1 Tag description

The message tag for the calendar is described in Table 2.46.

XML tag	Attribute	Required	Description
`calendar_get`	`-`	Yes	The calendar_get tag is used to get all the calendar tasks. See examples below.
	`task_id`	No	ID of the task. To get all the calendar tasks omit task_id.
`calendar_report`	`-`	Yes	The calendar_report tag is used to report all the calendar tasks. See examples below.
`calendar_set`	`-`	Yes	The calendar_set tag is used to set multiple the calendar tasks. See examples below.
`calendar_delete`	`-`	Yes	The calendar_delete tag is used to delete all the calendar tasks. See examples below.
	`task_id`	No	ID of the task. To delete all the calendar task omit task_id
`calendar_get_timezone`	`-`	Yes	This tag is used to request the timezone offset from a device
`calendar_set_timezone`	`-`	Yes	This tag is used the set the timezone offset on a device
	`offset`	Yes	The timezone offset
`calendar_report_timezone`	`-`	Yes	This tag is used to report the timezone offset from a device
	`offset`	Yes	The timezone offset
`calendar_get_memory`	`-`	Yes	This tag is used to request the calendar memory information from a device
`calendar_report_memory`	`-`	Yes	This tag is used to report the calendar memory information from a device
	`count`	Yes	The maximum number of calendar task the device supports
	`free_count`	Yes	The used number of calendar tasks

Table 2.46: Calendar tags

The child tags for the `calendar_report` and `calendar_set` message are described in Table 2.47.

XML tag	Attribute	Required	Description
`task`	`-`	Yes*	The task will execute a the specified date and time *Only required under calendar_report and calendar_set
	`task_id`	Yes	ID of the task
	`start`	Yes	The start date of the task in seconds since 01/01-1970.
	`action_id`	Yes	ID for the action that will be executed.
	`end`	No	The end date of the task in seconds since 01/01-1970
	`repeat`	No	When the task shall be repeated. The time is in seconds (Real number ex. 3600)
	`weekdays`	No	On which weekdays the task should be executed. This attribute is a bitmap, where bit 1 is monday and bit 7 is sunday. Bit 8 is not used.

Table 2.47: Calendar child tags

2.3.13.2 Examples

The following section contains XML examples for using the `calendar_get`, `calendar_report`, `calendar_set` and `calendar_delete` messages.

Get a specific calendar task

In order to get a specific calendar task, a `calendar_get` message has to be sent to the device with a task ID. See XML 2.64 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_get task_id="1"/>
  </device>
</network>
```

XML 2.64: Get the calendar task with task ID 1

Get all calendars

In order to get all calendar tasks, a `calendar_get` message has to be sent to the device without a task ID. See XML 2.65 for an example of the get all message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_get/>
  </device>
</network>
```

XML 2.65: Get all the calendar tasks

Report a specific calendar task

To report a specific calendar task, the `calendar_report` message has to be sent from the device with a task ID and its corresponding child tags as described earlier. See XML 2.66 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_report>
      <task task_id="1" start="1314976980" action_id="1" repeat="86400"/>
    </calendar_report>
  </device>
</network>
```

XML 2.66: Report the calendar task with ID 1

Report all calendar tasks

To report all calendar tasks, the `calendar_report` message has to be sent from the device with a task ID and its corresponding child tags as described earlier. See XML 2.67 for an example of the report all message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_report>
      <task task_id="1" start="1314976980" action_id="1" end="1314980580" repeat="3600"/>
      <task task_id="2" start="1314976980" action_id="2" repeat="604800"/>
    </calendar_report>
  </device>
</network>
```

XML 2.67: Report all calendar tasks

Set a specific calendar task

In order to set a specific calendar task, a `calendar_set` message has to be sent to the device with a task ID and its corresponding child tags as described earlier. See XML 2.68 for an example of the set message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_set>
      <task task_id="1" start="1314980580" action_id="1" repeat="86400"/>
    </calendar_set>
  </device>
</network>
```

XML 2.68: Set the calendar task with ID 1

Set multiple calendar tasks

In order to set multiple calendar tasks, a `calendar_set` message has to be sent to the device with a task ID ans its corresponding child tags as described earlier. See XML 2.69 for an example of the set multiple message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_set>
      <task task_id="1" start="1314976980" action_id="1" end="1314980580" repeat="3600"/>
      <task task_id="2" start="1314976980" action_id="2" repeat="604800"/>
    </calendar_set>
  </device>
</network>
```

XML 2.69: Set multiple calendar tasks

Delete a specific calendar task

In order to delete a specific calendar task, a `calendar_delete` message has to be sent to the device with a task ID. See XML 2.70 for an example of the delete message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_delete task_id="1"/>
  </device>
</network>
```

XML 2.70: Delete the calendar task with ID 1

Delete all calendar tasks

In order to delete all calendar tasks, a `calendar_delete` message has to be sent to the device without a task ID. See XML 2.71 for an example of the delete all message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_delete/>
  </device>
</network>
```

XML 2.71: Delete all calendar tasks

2.3.14 State machine

The state machine message is used to obtain a list of current state operations.

The state machine message is maintained by get, report, set and delete tags, as well as the state specific tags get_state and report_state. Each state machine is built from a list of valid states, and controlled by the transaction child tag. This transaction describes which action should be performed upon entering a given state, and to which state the state machine should go to next.

2.3.14.1 Tag description

The message tag for the state machine is described in Table 2.48.

XML tag	Attribute	Required	Description
statemachine_get	-	Yes	The statemachine_get tag is used to get states and transactions of the state machine. See examples below.
	statemachine_id	No	ID of the state machine. To get all the state machines omit statemachine_id.
statemachine_report	-	Yes	The statemachine_report tag is used to report states and transactions of the state machine. See examples below.
statemachine_set	-	Yes	The statemachine_set tag is used to set states and transactions of the state machine. See examples below.
statemachine_delete	-	Yes	The statemachine_delete tag is used to delete states and state machines. See examples below.
	statemachine_id	No	ID of the state machine. To delete all the state machines omit statemachine_id.
	state_id	No	ID of the state. To delete all the states omit state_id
statemachine_get_state	-	Yes	The statemachine_get_state tag is used to get the current state of the state machine. See examples below.
	statemachine_id	No	ID of the state machine. To get the current state of all the state machines omit statemachine_id.
statemachine_report_state	-	Yes	The statemachine_report_state tag is used to report the current state of the state machine. See examples below.
	statemachine_id	No	ID of the state machine. To report the current state for all the state machines omit statemachine_id.
statemachine_set_state	-	Yes	This tag is used to set the current state of the state machine. See examples below.
statemachine_get_memory	-	Yes	This tag is used to request the statemachine memory information from a device
statemachine_report_memory	-	Yes	This tag is used to report the statemachine memory information from a device
	count	Yes	The maximum number of statemachines the device supports
	free_count	Yes	The used number of statemachines
statemachine_get_state_memory	-	Yes	This tag is used to request the statemachine state memory information from a device
statemachine_report_state_memory	-	Yes	This tag is used to report the statemachine state memory information from a device
	count	Yes	The maximum number of statemachine states the device supports
	free_count	Yes	The used number of statemachine states

Table 2.48: State machine tags

The child tags for the statemachine_report and statemachine_set message are described in Table 2.49.

XML tag	Attribute	Required	Description
`statemachine`	`-`	Yes	The statemachine
	`statemachine_id`	Yes	ID of the state machine

Table 2.49: State machine message child tags

The child tags for the statemachine message are described in Table 2.50.

XML tag	Attribute	Required	Description
`state`	`-`	Yes	The state the state machine can enter.
	`state_id`	Yes	ID of the state. The state_id is only present when all the states are returned

Table 2.50: Statemachine child tags

The child tags for the state are described in Table 2.51.

XML tag	Attribute	Required	Description
`transaction`	`-`	Yes	Transaction is used for executing actions and changing states
	`calculation_id`	No	ID of the calculation that needs to be true for this transaction to be executed.
	`action_id`	No	ID of the action the state machine will execute when this transaction is executed
	`goto_state_id`	No	ID of the state the state machine will enter when this transaction is executed

Table 2.51: State child tags

2.3.14.2 Examples

The following section contains XML examples for using the "statemachine_get", "statemachine_report", "statemachine_set", "statemachine_delete", "statemachine_get_state" and "statemachine_report_state" messages.

Get a specific state machines specific state

In order to get a specific state from a specific state machine, a "statemachine_get" message has to be sent to the device with a state machine ID and a state ID. See XML 2.72 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_get statemachine_id="1" state_id="2"/>
  </device>
</network>
```

XML 2.72: Get the state machine with ID 1 and its state with ID 2

Get a specific state machine

In order to get a specific state machine, a "statemachine_get" message has to be sent to the device with a state machine ID and without state ID. See XML 2.73 for an example of the "get" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_get statemachine_id="1"/>
  </device>
</network>
```

XML 2.73: Get the complete state machine with ID 1

Get all state machines

In order to get all state machines, a "statemachine_get" message has to be sent to the device without a state machine ID and state ID. See XML 2.74 for an example of the "get all" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_get/>
  </device>
</network>
```

XML 2.74: Get all state machines

Report a specific state machines specific state

To report a specific state from a specific state machine, the "statemachine_report" message has to be sent from the device with a state machine ID on its corresponding child tags as described earlier. See XML 2.75 for an example of the "report" message.

Report a specific state machine

To report a specific state machine, the "statemachine_report" message has to be sent from the device with a state machine ID on its corresponding child tags as described earlier. See XML 2.76 for an example of the "report" message.

Report all state machines

To report all state machines, the "statemachine_report" message has to be sent from the device with the state machine ID on its corresponding child tags as described earlier. See XML 2.77 for an example of the "report all" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report>
      <statemachine statemachine_id="1">
        <state state_id="2">
          <transaction calculation_id="2" action_id="1" goto_state_id="3"/>
          <transaction calculation_id="1" action_id="2"/>
        </state>
      </statemachine>
    </statemachine_report>
  </device>
</network>
```

XML 2.75: Report the state machine with ID 1 and its state with ID 2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="2" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="2" action_id="1" goto_state_id="1"/>
          <transaction calculation_id="1" action_id="2"/>
        </state>
      </statemachine>
    </statemachine_report>
  </device>
</network>
```

XML 2.76: Report the state machines with ID 1 and all its states

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="2" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="2" action_id="1" goto_state_id="1"/>
          <transaction calculation_id="1" action_id="2"/>
        </state>
      </statemachine>
      <statemachine statemachine_id="2">
        <state state_id="1">
          <transaction calculation_id="3" action_id="4" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="4" action_id="4" goto_state_id="1"/>
          <transaction calculation_id="3" action_id="5"/>
        </state>
      </statemachine>
    </statemachine_report>
  </device>
</network>
```

XML 2.77: Report the state machines with ID 1 and the state machine with ID 2 and all there states

Set a specific state machines specific state

In order to set a specific state on a specific state machine, a "statemachine_set" message has to be sent to the device

with a state machine ID on its corresponding child tags as described earlier. See XML 2.78 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_set>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="2" goto_state_id="2"/>
        </state>
      </statemachine>
    </statemachine_set>
  </device>
</network>
```

XML 2.78: Set the state machine with ID 1 state 1

Set a specific state machine

In order to set a specific state machine, a "statemachine_set" message has to be sent to the device with a state machine ID on its corresponding child tags as described earlier. See XML 2.79 for an example of the "set" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_set>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="2" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="2" action_id="1" goto_state_id="1"/>
          <transaction calculation_id="1" action_id="2"/>
        </state>
      </statemachine>
    </statemachine_set>
  </device>
</network>
```

XML 2.79: Set multiple states in the state machine with ID 1

Set multiple state machines

In order to set multiple state machines, a "statemachine_set" message has to be sent to the device with the state machine ID on its corresponding child tags as described earlier. See XML 2.80 for an example of the "set multiple" message.

Delete a specific state machines specific state

In order to delete a specific state in a specific state machine, a "statemachine_delete" message has to be sent to the device with a state machine ID and state ID. See XML 2.81 for an example of the "delete" message.

Delete a specific state machine

In order to delete a specific state machine, a "statemachine_delete" message has to be sent to the device with a state machine ID and without state ID. See XML 2.82 for an example of the "delete" message.

Delete all state machines

In order to delete all state machines, a "statemachine_delete" message has to be sent to the device without a statemachine ID and state ID. See XML 2.83 for an example of the "delete all" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_set>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="2" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="2" action_id="1" goto_state_id="1"/>
          <transaction calculation_id="1" action_id="2"/>
        </state>
      </statemachine>
      <statemachine statemachine_id="2">
        <state state_id="1">
          <transaction calculation_id="3" action_id="4" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="4" action_id="4" goto_state_id="1"/>
          <transaction calculation_id="3" action_id="5"/>
        </state>
      </statemachine>
    </statemachine_set>
  </device>
</network>
```

XML 2.80: Set multiple states in the state machine with ID 1 and 2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_delete statemachine_id="1" state_id="2"/>
  </device>
</network>
```

XML 2.81: Delete the state with ID 2 from the state machine with ID 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_delete statemachine_id="1"/>
  </device>
</network>
```

XML 2.82: Delete the state machines with ID 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_delete/>
  </device>
</network>
```

XML 2.83: Delete all state machines

Get the current state

In order to get the current state from a state machine, a "statemachine_get_state" message has to be sent to the device with a state machine ID. See XML 2.84 for an example of the "get" message.

Report the current state

To report the current state of a state machine, the "statemachine_report_state" message has to be sent from the

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_get_state statemachine_id="1"/>
  </device>
</network>
```

XML 2.84: Get the current state of the state machine

device with a state machine ID and its current state. See XML 2.85 for an example of the "report" message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report_state>
      <statemachine_state statemachine_id="1">3</statemachine_state>
    </statemachine_report_state>
  </device>
</network>
```

XML 2.85: Report the current state of the state machine

2.3.15 Firmware update

The firmware update is used to update the firmware on the devices.

2.3.15.1 Tag description

The message tag for the firmware update is described in Table 2.52.

XML tag	Attribute	Required	Description
`firmware_init`	-	Yes	Initialize the firmware update procress.
	`size`	Yes	The size of the new firmware
	`firmware_id`	Yes	A id of the new firmware
	`checksum`	Yes	The checksum of the new firmware
`firmware_data`	-	Yes	Sends a chunk of the firmware image
	`offset`	Yes	The offset in the firmware image
`firmware_update_start`	-	Yes	Starts the firmware update, if the check sum matches
`firmware_report`	-	Yes	Reports the status of the last firmware update message
	`status`	Yes	The status of last message
	`expected_offset`	No	The expected offset of the next firmware data message
`firmware_information_get`	-	Yes	Requests a firmware information report
`firmware_information_report`	-	Yes	Reports the current information about the firmware
	`size`	Yes	The size of the firmware
	`firmware_id`	Yes	The firmware ID
	`received_size`	Yes	The number of bytes received
	`chunk_size`	Yes	The size of the chunks in firmware data

Table 2.52: Firmware update tags

The child tags for the *firmware_data* message are described in Table 2.53.

XML tag	Attribute	Required	Description
`chunk`	-	Yes	A chunk of the firmware image

Table 2.53: Firmware_data message child tags

Status

The valid options for the *status* attribute in the *firmware_report* are listed in Table 2.54.

Status Id	Status Type
1	OK
2	Not Initialized
3	Size is to big
4	Chechsum Error in received data
5	Data Overflow
6	Wrong Offset
7	Chunk size is to big
8	Data is missing
9	Chunk size is to small

Table 2.54: Firmware Update Status Type

2.3.15.2 Examples

The following section contains XML examples for using the *firmware_update* messages.

Initialize the firmware

In order to initialize the firmware, a *firmware_init* message has to be sent to the device with a size and a firmware ID. See XML 2.86 for an example of the init message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="firmware_update.xsd">
  <device version="1">
    <firmware_init size="4000" firmware_id="1" checksum="12AB"/>
  </device>
</network>
```

XML 2.86: Initialize the firmware

Send a chunk of the firmware image

In order to send a chunk of the firmware image, a *firmware_data* message has to be sent to the device with an offset and its corresponding child tags as described earlier. See XML 2.87 for an example of the data message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="firmware_update.xsd">
  <device version="1">
    <firmware_data offset="256">
      <chunk>73652847352A8464B8465C9745F4</chunk>
    </firmware_data>
  </device>
</network>
```

XML 2.87: A chunk of the firmware image

Start the firmware update

In order to start the firmware update, a *firmware_update_start* message has to be sent to the device with a checksum. See XML 2.88 for an example of the start message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="firmware_update.xsd">
  <device version="1">
    <firmware_update_start/>
  </device>
</network>
```

XML 2.88: Start firmware update

Report the status of the last firmware update

In order to report the status of the last firmware update, a *firmware_report* message has to be sent from the device with a status and an expected offset. See XML 2.89 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="firmware_update.xsd">
  <device version="1">
    <firmware_report status="0" expected_offset="256"/>
  </device>
</network>
```

XML 2.89: Firmware report

Get the firmware information

In order to get the firmware information, a *firmware_information_get* message has to be sent to the device. See XML 2.90 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="firmware_update.xsd">
  <device version="1">
    <firmware_information_get/>
  </device>
</network>
```

XML 2.90: Get the firmware information

Report the firmware information

In order to report the firmware information, a *firmware_information_report* message has to be sent from the device with a size, a firmware ID, a received size and a chunk size. See XML 2.91 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="firmware_update.xsd">
  <device version="1">
    <firmware_information_report size="4000" firmware_id="1" received_size="512" chunk_size="256"/>
  </device>
</network>
```

XML 2.91: Firmware information report

2.3.16 Status

The status is used to report information from a device.

2.3.16.1 Tag description

The message tag for the status is described in Table 2.55.

XML tag	Attribute	Required	Description
`status_report`	-	Yes	This tag is used to report a status
	`type_id`	Yes	The type of status. See Table 2.56
	`code`	Yes	The code for the status
	`level`	Yes	The level of the status. See Table 2.57
	`data`	No	Optimal data for the status
`status_get_level`	-	Yes	This tag is used to request the status level from a device
`status_set_level`	-	Yes	This tag is used to set the status level on a device
	`level`	Yes	The new status level for a the device. See Table 2.57
`status_report_level`	-	Yes	This tag is used to report the status level from a device
	`level`	Yes	The new status level for a the device. See Table 2.57

Table 2.55: Status tags

The valid types for the status is described in Table 2.56.

Id	Type
1	Public key
2	Memory Information
3	Device Description
4	Value Description
5	Value
6	Partner Information
7	Action
8	Calculation
9	Timer
10	Calendar
11	State machine
12	Firmware update
13	Configuration
100	Exi
101	System
200	Application

Table 2.56: Status type

The valid level for the status level is described in Table 2.57.

Level	Name	Description
0	Disabled	No status reports will be send
1	Fatal	Fatal errors that can't be recovered from
2	Error	An error
3	Warning	A warning
4	Info	Very verbose information
5	Debug	Debug information

Table 2.57: Status level

The device description code is described in Table 2.58.

Code	Name	Description
11	Set wrong ID	Trying to set a read only value
12	Wrong size of Channel Map	The channel map must have 4 channels
13	Missing Synchrozations channels	The channel map is missing the synchrozations channels

Table 2.58: Status device description code

The value description code is described in Table 2.59.

Code	Name	Description
1	Get Wrong ID	Trying to get a value description with a invalid id
2	Set Wrong ID	Trying to add a virtual value description with a invalid id
3	Delete Wrong ID	Trying to delete a virtual value with a invalid id
11	Not Supported	Trying to add a virtual value with a type that is not supported
12	Invalid Step	Trying to add a virtual value with a invalid step

Table 2.59: Status value description code

The value code is described in Table 2.60.

Code	Name	Description
1	Get Wrong ID	Trying to get a value with a invalid id
2	Set Wrong ID	Trying to set a value with a invalid id
11	Check Wrong ID	Trying to use a value with a wrong id
12	Value Invalid	The value is invalid
13	Wrong Data Type	Trying to set a value with a wrong data type
14	Invalid Step	The step is not supported
15	Can't read write only	Trying to read a write only value
16	Can't write read only	Trying to write a read only value

Table 2.60: Status value code

The partner information code is described in Table 2.61.

Code	Name	Description
1	Get wrong Id	Trying to get a partner that is not configured
2	Set wrong Id	Trying to set a partner with a wrong Id
3	Delete wrong Id	Trying to delete a partner that is not configured
11	Wrong Id	The reference partner is not configured
12	Failed to send to partner	Sending a meesage to a partner but there was no reply
13	Set wrong type	Trying set a partner with an unsupported info type

Table 2.61: Status partner information code

The action code is described in Table 2.62.

Code	Name	Description
1	Get wrong Id	Trying to get an action that is not configured
2	Set wrong Id	Trying to set an action with an invalid id
3	Delete wrong Id	Trying to delete an action that is not configured
11	Execute wrong Id	Trying to execute an action that is not configured
12	Execute queue overflow	Trying to enqueue action, but queue is full

Table 2.62: Status action code

The calculation code is described in Table 2.63.

Code	Name	Description
1	Get wrong Id	Trying to get an calculation that is not configured
2	Set wrong Id	Trying to set an calculation with an invalid id
3	Delete wrong Id	Trying to delete an calculation that is not configured
11	Check wrong Id	Trying to evaluate a calculation that is not configured

Table 2.63: Status calculation code

The timer code is described in Table 2.64.

Code	Name	Description
1	Get wrong Id	Trying to get a timer that is not configured
2	Set wrong Id	Trying to set a timer with a wrong id
3	Delete wrong Id	Trying to delete a timer that is not configured
11	Start wrong Id	Trying to start a timer that is not configured
12	Stop wrong Id	Trying to stop a timer that is not configured

Table 2.64: Status timer code

The calendar code is described in Table 2.65.

Code	Name	Description
1	Get wrong Id	Trying to get a calendar task that is not configured
2	Set wrong Id	Trying to set a calendar task with a wrong id
3	Delete wrong Id	Trying to delete a calendar task that is not configured

Table 2.65: Status calendar code

The state machine code is described in Table 2.66.

Code	Name	Description
1	Get wrong Id	Trying to get a statemachine with an invalid id
2	Set wrong Id	Trying to set a statemachine with an invalid id
3	Delete wrong Id	Trying to delete a statemachine with an invalid id
4	Get wrong state Id	Trying to get a statemachine state with an invalid id
5	Set wrong state Id	Trying to set a statemachine state with an invalid id
6	Delete wrong state Id	Trying to delete a statemachine state with an invalid id
11	Check wrong id	Trying to get a statemachine state with invalid id
12	Running too long	The execution of the statemachine was been running too long

Table 2.66: Status state machine code

The firmware update code is described in Table 2.67.

Code	Name	Description
11	Failed to upgrade	Failed to upgrade firmware from bootloader

Table 2.67: Status firmware update code

The configuration code is described in Table 2.68.

Code	Name	Description
11	Timeout	The started configuration timed out and the configuration is rolled back
12	Invalid	The configuration saved on the device is invalid and needs to be valided
13	Started	The configuration has started on the device and the statemachine is stopped

Table 2.68: Status configuration code

The system code is described in Table 2.69.

Code	Name	Description
11	No NTP	Failed to synchronize with the NTP Server
12	Awake	Device was been woken and is now awake

Table 2.69: Status system code

2.3.16.2 Examples

This flowering section consists of examples in xml for how to use the *status_report*, *status_get_level*, *status_set_level* and *status_report_level* message.

Status report

To report status, a *status_report* message has to be sent from the device. See XML 2.92 for an example of the *report* message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="status.xsd">
  <device version="1">
    <status_report type_id="7" code="1" level="2"/>
  </device>
</network>
```

XML 2.92: Status report

Status get level

In order to get the status level, a *status_get_level* message has to be sent from the device. See XML 2.93 for an example of the *get level* message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="status.xsd">
  <device version="1">
    <status_get_level/>
  </device>
</network>
```

XML 2.93: Status get level

Status report level

To report the status level, a *status_report_level* message has to be sent from the device. See XML 2.94 for an example of the *report level* message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="status.xsd">
  <device version="1">
    <status_report_level level="2"/>
  </device>
</network>
```

XML 2.94: Status report level

Status set level

In order to set the the status level, a *status_set_level* message has to be sent to the device. See XML 2.95 for an example of the *set level* message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="status.xsd">
  <device version="1">
    <status_set_level level="3"/>
  </device>
</network>
```

XML 2.95: Set status level

2.3.17 Configuration

The configuration service is used to save the configuration on a device or discard it.

2.3.17.1 Tag description

The message tag for the configuration is described in Table 2.70.

XML tag	Attribute	Required	Description
`config_status_get`	`-`	Yes	This tag is used to request the configuration status from a device
`config_status_report`	`-`	Yes	This tag is used to report the configuration status
	`status`	Yes	The configuration status. See Table 2.71
`config_mode_set`	`-`	Yes	This tag is used to set the configuration mode on a device
	`mode`	Yes	The configuration mode. See Table 2.72

Table 2.70: Configuration tags

The valid values for the configuration status is described in Table 2.71.

Status	Name	Description
0	Idle	When there is not configuration started
1	Started	When configuration is started

Table 2.71: Configuration Status

The valid configuration modes is described in Table 2.72.

Status	Name	Description
0	Rollback	Rollback any configuration changes
1	Save and Reset	Save any configuration changes and reset the statemachine states
1	Save and Preserve	Save any configuration changes and do not change the statemachine states
1	Set Default	Clear the configuration and set the default configuration
1	Clear	Clear the configuration

Table 2.72: Configuration Mode

2.3.17.2 Examples

This flowering section consists of examples in xml for how to use the *configuration_status_get*, *configuration_status_report* and *configuration_mode_set* message.

Get configuration status

In order to get the configuration status, a *configuration_status_get* message has to be sent to the device. See XML 2.96 for an example of the get message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="configuration.xsd">
  <device version="1">
    <config_status_get/>
  </device>
</network>
```

XML 2.96: Get configuration status

Report configuration status

To report the configuration status, a *configuration_status_report* message has to be sent from the device. See XML 2.97 for an example of the report message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="configuration.xsd">
  <device version="1">
    <config_status_report status="0"/>
  </device>
</network>
```

XML 2.97: Report configuration status

Set configuration mode

In order to set the the configuration mode, a *configuration_mode_set* message has to be sent to the device. See XML 2.98 for an example of the mode message.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="configuration.xsd">
  <device version="1">
    <config_mode_set mode="1"/>
  </device>
</network>
```

XML 2.98: Set configuration mode

2.4 User stories

In this chapter the different user stories will be described. In every section there are descriptions of the user stories a drawing and examples of how to describe the scenario in xml.

2.4.1 Remote Control with wall switch controlling multiple devices

An individual from the household comes home from work and wants to turn on both the light and the TV.

He/she pushes the wall switch (Wall switch) connected to the electrical outlets of both the light and the TV, which subsequently turns both these appliances on.

Later that evening, when he/she is getting ready for bed, he/she again pushes the wall switch, which in turn shuts off all appliances connected to this switch, i.e. the lights and the TV.

2.4.1.1 Illustration

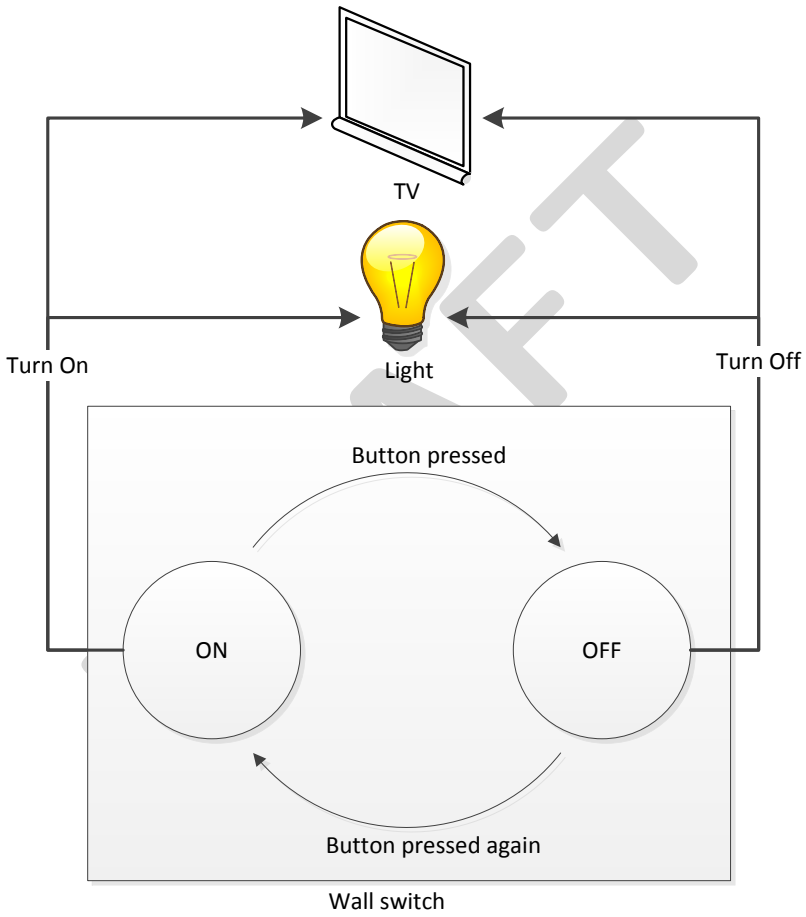

Figure 2.5: Remote control with light and TV

2.4.1.2 XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_report>
      <calculation calculation_id="1" method_id="6">
        <left value_id="1"/>
        <right constant_number="1"/>
      </calculation>
      <calculation calculation_id="1" method_id="6">
        <left value_id="1"/>
        <right constant_number="0"/>
      </calculation>
    </calculation_report>
  </device>
</network>
```

XML 2.99: Calculation for User Story 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <set value_id="1" partner_id="1" number="0"/>
        <set value_id="1" partner_id="2" number="0"/>
      </action>
      <action action_id="2">
        <set value_id="1" partner_id="1" number="1"/>
        <set value_id="1" partner_id="2" number="1"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.100: Action for User Story 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="1" goto_state_id="2"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="2" action_id="2" goto_state_id="1"/>
        </state>
      </statemachine>
    </statemachine_report>
  </device>
</network>
```

XML 2.101: State Machine for User Story 1

2.4.2 Temperature Control with calendar tasks

The temperature control in the house is set to 16 degrees Celsius at night-time.

An individual wakes up at 6:00 every morning, and wants the house to have an ambient temperature of 19 degrees Celsius. Later when he/she leaves for work at 8:00, the temperature should return to a steady temperature of 16 degrees Celsius.

In the late afternoon, when he/she comes home at 18:00, the temperature should be at an ambient temperature of 21 degrees Celsius, followed by a decrease in temperature steadying at 16 degrees when going to bed at 22:00 in the evening.

2.4.2.1 Illustration

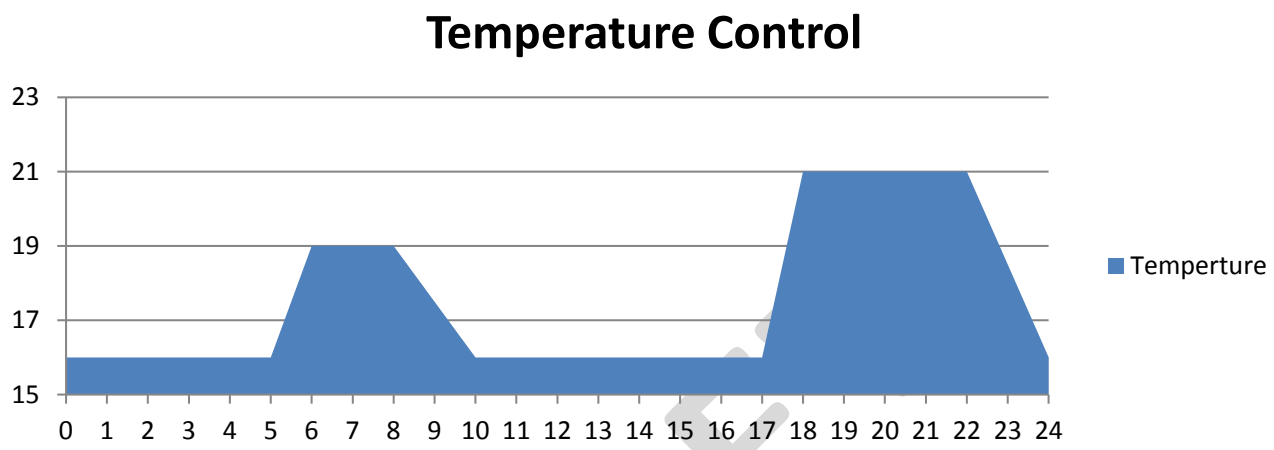

Figure 2.6: Temperature control with calendar tasks

2.4.2.2 XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <set value_id="1" number="16"/>
      </action>
      <action action_id="2">
        <set value_id="1" number="19"/>
      </action>
      <action action_id="3">
        <set value_id="1" number="21"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.102: Action for User Story 2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_report>
      <task task_id="1" start="1319432400" repeat="86400"/>
      <task task_id="2" start="1319443200" repeat="86400"/>
      <task task_id="3" start="1319475600" repeat="86400"/>
      <task task_id="4" start="1319493600" repeat="86400"/>
    </calendar_report>
  </device>
</network>
```

XML 2.103: Calendar for User Story 2

2.4.3 Temperature Control with window and temperature sensor

The temperature in the house is at a steady 21 degrees Celsius.

An individual from the household opens a window in a room, which is detected by the system. The thermostat detects, a possible, decrease in room temperature, but allows the temperature to drop because of the opened windows.

After 5 minutes he/she closes the window again, and the temperature should rise and steady at the before mentioned 21 degrees Celsius.

2.4.3.1 Illustration

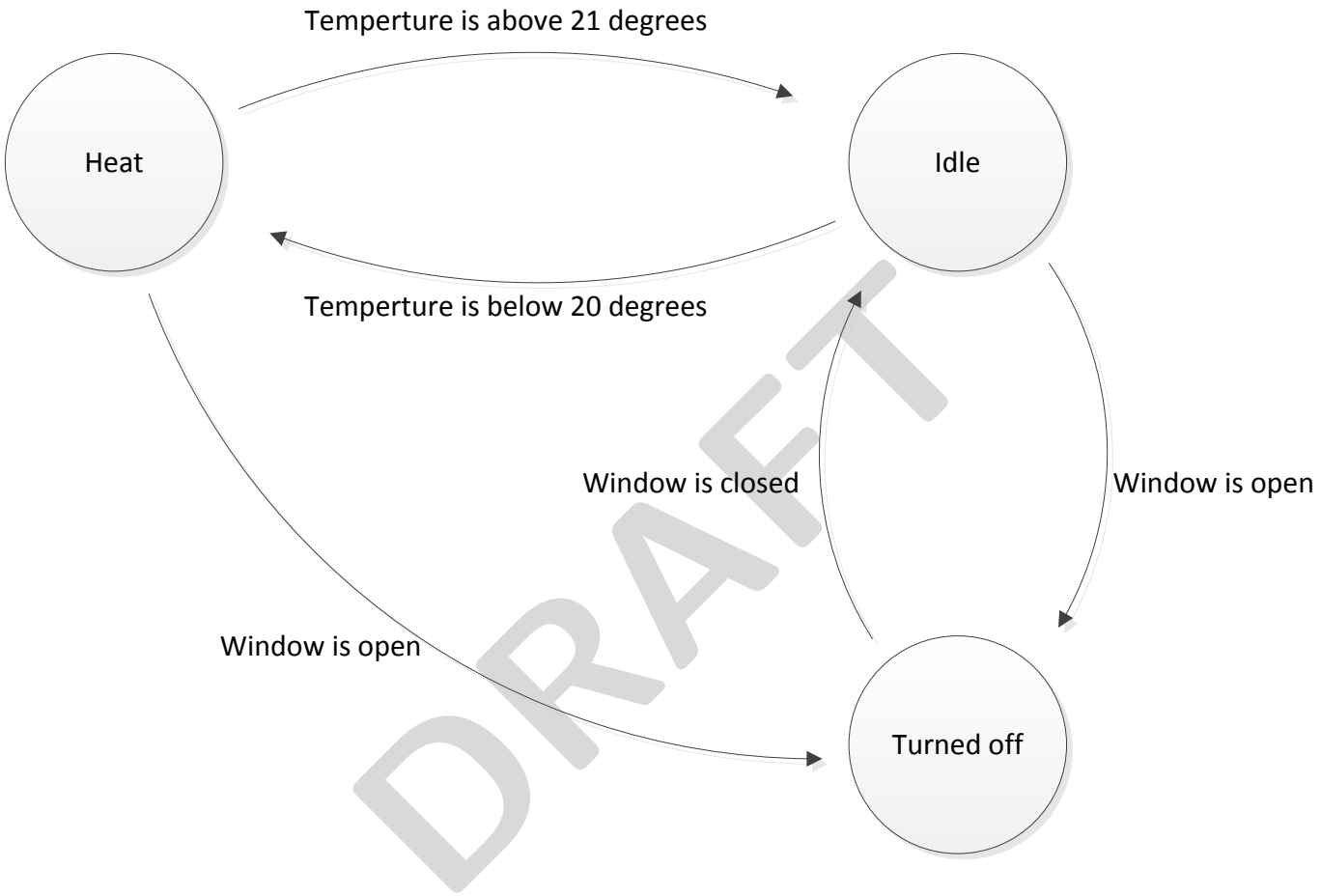

Figure 2.7: Temperature control with window and temperature sensor

2.4.3.2 XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_report>
      <calculation calculation_id="1" method_id="9">
        <!-- Partner 2 is temperature -->
        <left value_id="1" partner_id="2"/>
        <right constant_number="21"/>
      </calculation>
      <calculation calculation_id="2" method_id="8">
        <left value_id="1" partner_id="2"/>
        <right constant_number="20"/>
      </calculation>
      <calculation calculation_id="3" method_id="6">
        <!-- Partner 3 is window -->
        <left value_id="1" partner_id="3"/>
        <right constant_number="0"/>
      </calculation>
      <calculation calculation_id="4" method_id="6">
        <left value_id="1" partner_id="3"/>
        <right constant_number="1"/>
      </calculation>
    </calculation_report>
  </device>
</network>
```

XML 2.104: Calculation for User Story 3

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <set value_id="1" number="1"/>
      </action>
      <action action_id="2">
        <set value_id="1" number="0"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.105: Action for User Story 3

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="2" action_id="1" goto_state_id="2"/>
          <transaction calculation_id="4" action_id="2" goto_state_id="3"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="1" action_id="2" goto_state_id="1"/>
          <transaction calculation_id="4" action_id="2" goto_state_id="3"/>
        </state>
        <state state_id="3">
          <transaction calculation_id="3" goto_state_id="1"/>
        </state>
      </statemachine>
    </statemachine_report>
  </device>
</network>
```

XML 2.106: State Machine for User Story 3

2.4.4 Light control with movement and luminance sensor

At day-time when it is light outside, an individual walks into the living room. This movement is detected by the system, but the lights are not turned on as it is not necessary.

Later that evening, when it has turned dark outside, he/she walks into the living room again, turning the lights on in this room.

He/she leaves the living room, and 5 minutes after the lights are turned off again, due to the system not picking up any movement in the room for the 5 minute duration.

2.4.4.1 Illustration

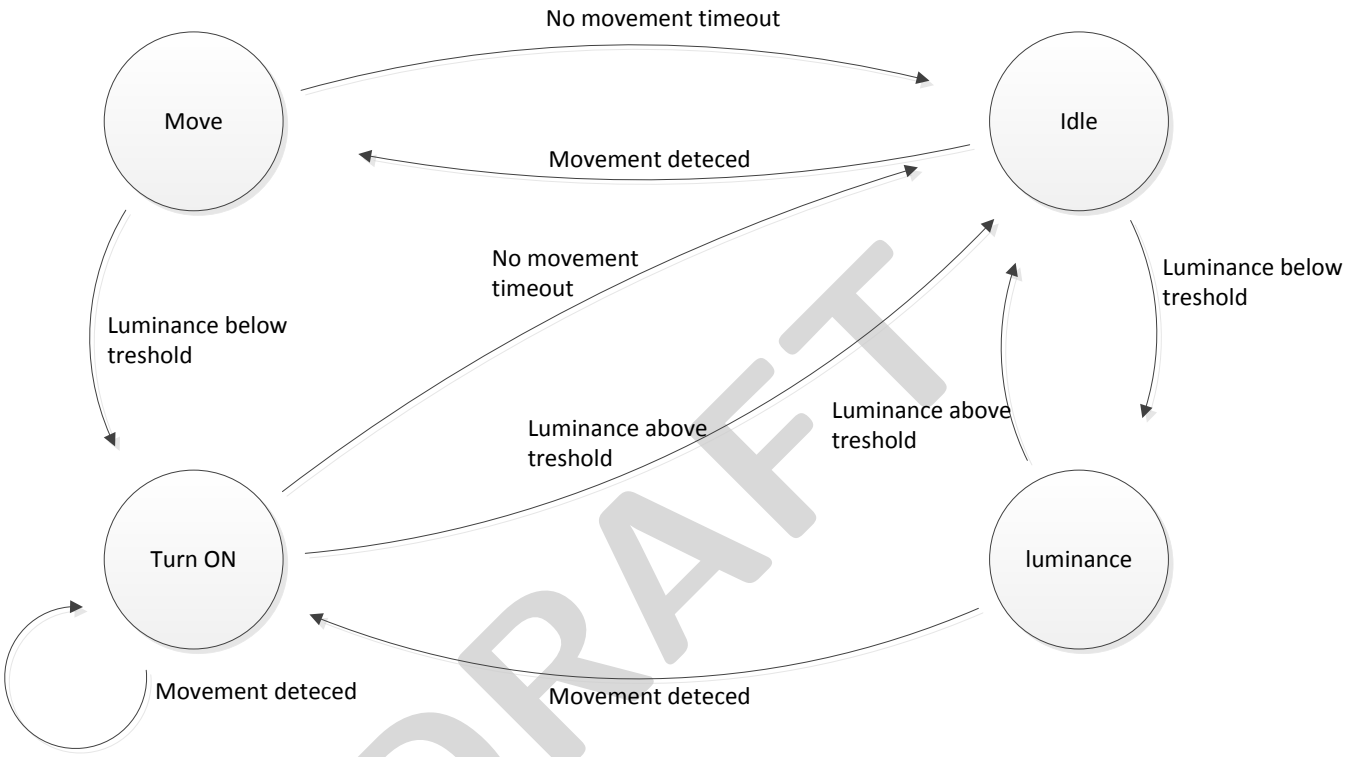

Figure 2.8: Light control with movement and luminance sensor

2.4.4.2 XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calculation.xsd">
  <device version="1">
    <calculation_report>
      <calculation calculation_id="1" method_id="6">
        <!-- Partner 2 is movement -->
        <left value_id="1" partner_id="2"/>
        <right constant_number="1"/>
      </calculation>
      <calculation calculation_id="2" method_id="8">
        <!-- Partner 3 is luminance -->
        <left value_id="1" partner_id="3"/>
        <right constant_number="50"/>
      </calculation>
      <calculation calculation_id="3" method_id="9">
        <left value_id="1" partner_id="3"/>
        <right constant_number="75"/>
      </calculation>
      <calculation calculation_id="4" method_id="6">
        <left timer_id="1"/>
        <right constant_number="1"/>
      </calculation>
    </calculation_report>
  </device>
</network>
```

XML 2.107: Calculation for User Story 4

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <set value_id="1" number="0"/>
      </action>
      <action action_id="2">
        <set value_id="1" number="0"/>
      </action>
      <action action_id="3">
        <timer_start timer_id="1"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.108: Action for User Story 4

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="statemachine.xsd">
  <device version="1">
    <statemachine_report>
      <statemachine statemachine_id="1">
        <state state_id="1">
          <transaction calculation_id="1" action_id="3" goto_state_id="2"/>
          <transaction calculation_id="2" goto_state_id="3"/>
        </state>
        <state state_id="2">
          <transaction calculation_id="2" action_id="1" goto_state_id="4"/>
          <transaction calculation_id="4" goto_state_id="1"/>
        </state>
        <state state_id="3">
          <transaction calculation_id="3" goto_state_id="1"/>
          <transaction calculation_id="1" action_id="1" goto_state_id="4"/>
        </state>
        <state state_id="4">
          <transaction calculation_id="3" action_id="2" goto_state_id="1"/>
          <transaction calculation_id="1" goto_state_id="3"/>
          <transaction calculation_id="4" action_id="2" goto_state_id="1"/>
        </state>
      </statemachine>
    </statemachine_report>
  </device>
</network>
```

XML 2.109: State Machine for User Story 4


```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="timer.xsd">
  <device version="1">
    <timer_report>
      <execute timer_id="1" after="300000"/>
    </timer_report>
  </device>
</network>
```

XML 2.110: Timer for User Story 4

2.4.5 Washing machine control

An individual prepares the washing machine by putting in the clothing and applying laundry detergent and possibly fabric softener if needed.

He/she then leaves for work, but wants the washing machine to be turned on, so that it will have finished is laundry cycle when he/she comes home from work.

He/she sets the washing program to wool, and tells the system that he/she will be home at 18:00. The system determines the best time for the washing machine to start for it to be finished when the user comes home from work.

Later that day, he/she finds that they will be coming home earlier and starts the washing machine manually, thereby overruling the starting time set by the system.

2.4.5.1 Illustration

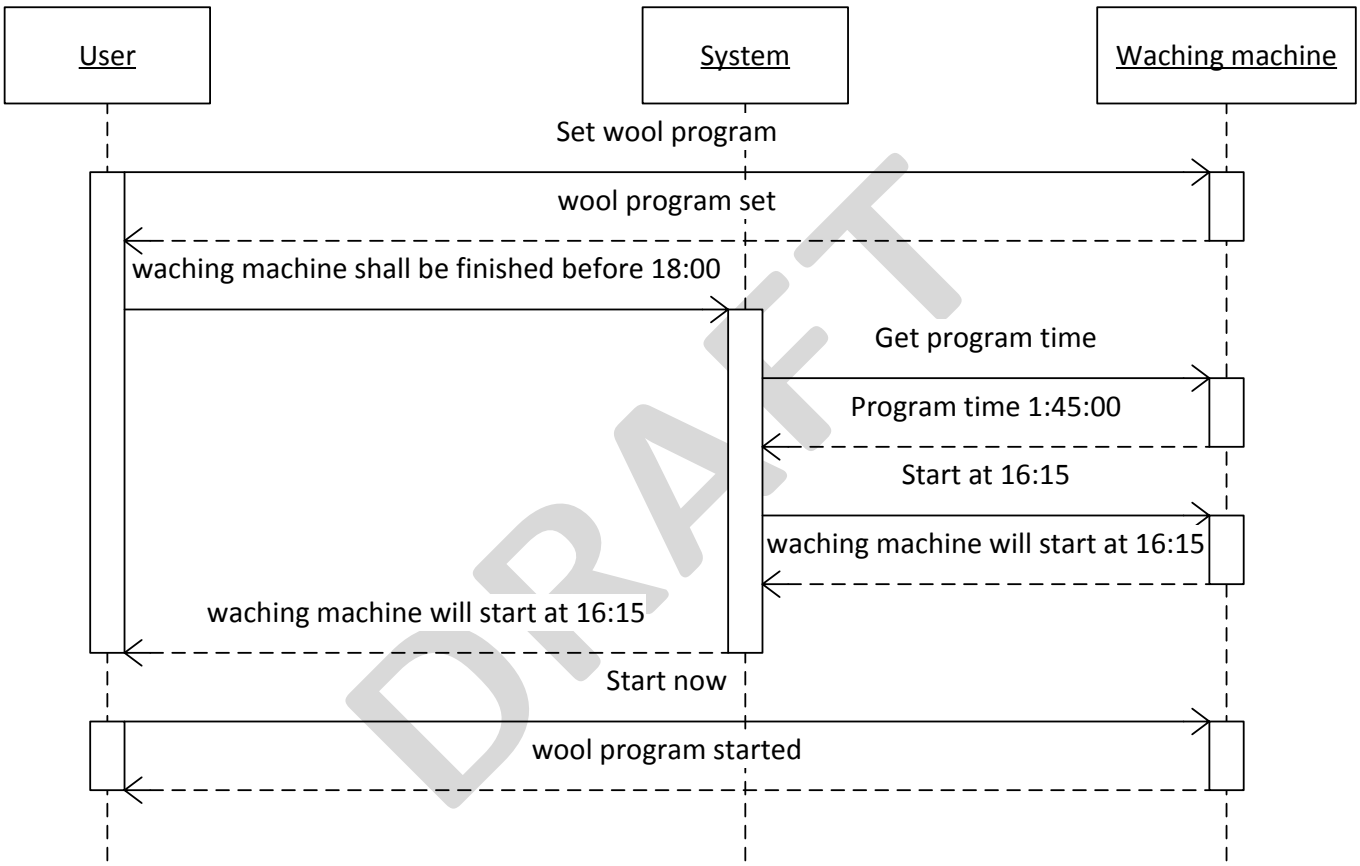

Figure 2.9: Washing machine control

2.4.5.2 XML

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="value.xsd">
  <device version="1">
    <value_set value_id="1" timestamp="0" string="wool"/>
    <value_get value_id="2"/>
    <value_report value_id="2" timestamp="0" string="01:45:00"/>
    <value_set value_id="1" timestamp="0" string="start"/>
  </device>
</network>
```

XML 2.111: Value for User Story 5

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <set value_id="3" string="start"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.112: Action for User Story 5

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_report>
      <task task_id="1" start="1319472900" action_id="1"/>
    </calendar_report>
  </device>
</network>
```

XML 2.113: Calendar for User Story 5

2.4.6 Electricity pricing

The provider of electricity to the household, i.e. the electricity company, sends pricings to the electric meter on a regular daily basis.

This information is used to reduce the electricity bill. This reduction is done by performing actions that have a larger cost, in periods with low pricing, e.g. at night-time or mid-day on workdays.

2.4.6.1 Illustration

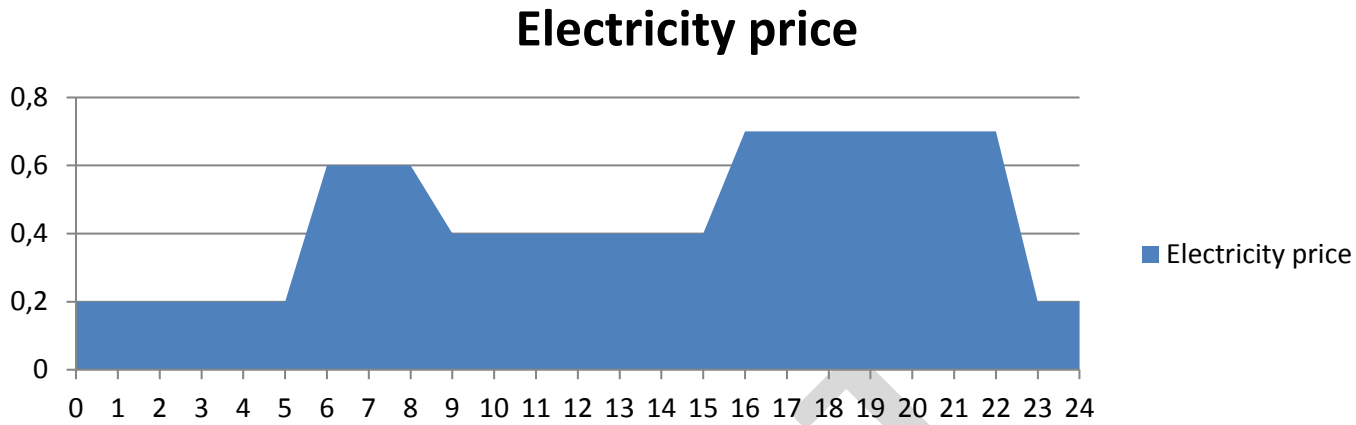

Figure 2.10: Electricity pricing

2.4.6.2 XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="action.xsd">
  <device version="1">
    <action_report>
      <action action_id="1">
        <set value_id="1" number="0.2"/>
      </action>
      <action action_id="2">
        <set value_id="1" number="0.4"/>
      </action>
      <action action_id="3">
        <set value_id="1" number="0.6"/>
      </action>
      <action action_id="4">
        <set value_id="1" number="0.7"/>
      </action>
    </action_report>
  </device>
</network>
```

XML 2.114: Action for User Story 6

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1"
    xsi:noNamespaceSchemaLocation="calendar.xsd">
  <device version="1">
    <calendar_report>
      <task task_id="1" start="1319436000" action_id="3" end="1319446800" repeat="86400"/>
      <task task_id="2" start="1319446800" action_id="2" end="1319472000" repeat="86400"/>
      <task task_id="3" start="1319472000" action_id="4" end="1319497200" repeat="86400"/>
      <task task_id="4" start="1319497200" action_id="1" end="1319522400" repeat="86400"/>
    </calendar_report>
  </device>
</network>
```

XML 2.115: Calendar for User Story 6

List of XSD's

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:simpleType name="payloadType">
    <xs:restriction base="xs:hexBinary">
      <xs:minLength value="0"/>
      <xs:maxLength value="2047"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="phy">
    <xs:complexType>
      <xs:attribute name="payload" type="payloadType" use="required"/>
      <xs:attribute name="phy_layer_version" type="xs:unsignedInt" use="required"/>
      <xs:attribute name="security" type="xs:unsignedInt" use="required"/>
      <xs:attribute name="foward_error_correction" type="xs:unsignedInt" use="required"/>
      <xs:attribute name="foward_error_correction_length" type="xs:unsignedInt" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 116: Phy XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:simpleType name="frame_nonceType">
    <xs:restriction base="xs:hexBinary">
      <xs:length value="6"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="mac_source_addressType">
    <xs:restriction base="xs:hexBinary">
      <xs:minLength value="2"/>
      <xs:maxLength value="17"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="frame_integrity_codeType">
    <xs:restriction base="xs:hexBinary">
      <xs:length value="4"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="mac_destination_adressType">
    <xs:restriction base="xs:hexBinary">
      <xs:minLength value="2"/>
      <xs:maxLength value="17"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="mapType">
    <xs:restriction base="xs:hexBinary">
      <xs:length value="4"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="mac_option_ack_requestType">
    <xs:attribute name="nr_retries" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="mac_option_ackType">
    <xs:attribute name="rssi" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="mac_option_fragmentType">
    <xs:attribute name="is_last" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="offset" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="mac_option_channel_mapType">
    <xs:attribute name="type" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="map" type="mapType" use="required"/>
  </xs:complexType>
  <xs:complexType name="mac_option_wake_on_radioType">
    <xs:attribute name="timestamp" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="fraction" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="interval" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="channel" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="mac">
    <xs:complexType>
```

XML 117: Mac XSD 1/2

```
        <xs:element name="mac_option_ack_request" type="mac_option_ack_requestType" minOccurs="1"
            maxOccurs="1"/>
        <xs:element name="mac_option_ack" type="mac_option_ackType" minOccurs="1" maxOccurs="1"/>
        <xs:element name="mac_option_fragment" type="mac_option_fragmentType" minOccurs="0"
            maxOccurs="1"/>
        <xs:element name="mac_option_channel_map" type="mac_option_channel_mapType" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="mac_option_wake_on_radio" type="mac_option_wake_on_radioType" minOccurs="1"
            maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="mac_layer_version" type="xs:unsignedInt" use="required"/>
      <xs:attribute name="frame_nonce" type="frame_nonceType" use="required"/>
      <xs:attribute name="mac_source_address" type="mac_source_addressType" use="required"/>
      <xs:attribute name="frame_integrity_code" type="frame_integrity_codeType" use="required"/>
      <xs:attribute name="mac_destination_adress" type="mac_destination_adressType" use="required"/>
      <xs:attribute name="content_type" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 118: Mac XSD 2/2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="network_include">
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
        <xs:attribute name="address_size" type="xs:unsignedByte" use="optional"/>
        <xs:attribute name="inclusion_count" type="xs:unsignedInt" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="1" maxOccurs="1">
              <xs:element name="network_include" type="network_include"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 119: Network Management XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="publicKeyGetType">
  </xs:complexType>
  <xs:complexType name="publicKeyReportType">
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
        <xs:attribute name="key_type" type="xs:unsignedInt" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="1">
              <xs:element name="publickey_get" type="publicKeyGetType"/>
              <xs:element name="publickey_report" type="publicKeyReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 120: Public Key XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="serviceDescriptionGetType">
  </xs:complexType>
  <xs:complexType name="serviceType">
    <xs:attribute name="service_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="serviceDescriptionReportType">
    <xs:sequence>
      <xs:element name="service" type="serviceType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="1">
              <xs:element name="service_description_get" type="serviceDescriptionGetType"/>
              <xs:element name="service_description_report" type="serviceDescriptionReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 121: Service Description XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="memoryInformationGetType">
  </xs:complexType>
  <xs:complexType name="memoryInformationType">
    <xs:attribute name="memory_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="memoryInformationReportType">
    <xs:sequence>
      <xs:element name="memory_information" type="memoryInformationType" minOccurs="0"
    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="memory_information_get" type="memoryInformationGetType"/>
              <xs:element name="memory_information_report" type="memoryInformationReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 122: Memory Information XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="infoType">
    <xs:attribute name="type_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="number" type="xs:unsignedLong" use="optional"/>
    <xs:attribute name="string" type="xs:string" use="optional"/>
    <xs:attribute name="hex" type="xs:hexBinary" use="optional"/>
  </xs:complexType>
  <xs:complexType name="deviceDescriptionGetType">
  </xs:complexType>
  <xs:complexType name="deviceDescriptionType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="info" type="infoType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="device_description_get" type="deviceDescriptionGetType"/>
              <xs:element name="device_description_report" type="deviceDescriptionType"/>
              <xs:element name="device_description_set" type="deviceDescriptionType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 123: Device Description XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="numberFormatType">
    <xs:attribute name="unit" type="xs:string" use="required"/>
    <xs:attribute name="min" type="xs:double" use="required"/>
    <xs:attribute name="max" type="xs:double" use="required"/>
    <xs:attribute name="step" type="xs:double" use="required"/>
  </xs:complexType>
  <xs:complexType name="stringFormatType">
    <xs:sequence>
      <xs:element name="valid_value" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="max_length" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="hexBinaryFormatType">
    <xs:attribute name="max_length" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="valueDescriptionGetType">
    <xs:attribute name="value_description_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="valueDescriptionType">
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="number_format" type="numberFormatType"/>
      <xs:element name="string_format" type="stringFormatType"/>
      <xs:element name="hexBinary_format" type="hexBinaryFormatType"/>
    </xs:choice>
    <xs:attribute name="value_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="type_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="mode" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="persistent" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="min_log_interval" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="max_log_values" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="virtual" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="valueDescriptionReportType">
    <xs:sequence>
      <xs:element name="value_description" type="valueDescriptionType" minOccurs="0"
    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="valueDescriptionAddType">
    <xs:sequence>
      <xs:element name="value_description" type="valueDescriptionType" minOccurs="0"
    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="valueDescriptionDeleteType">
    <xs:attribute name="value_description_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="valueDescriptionMemoryGetType">
  </xs:complexType>
```

XML 124: Value Description XSD 1/2

```
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="value_description_get" type="valueDescriptionGetType"/>
              <xs:element name="value_description_report" type="valueDescriptionReportType"/>
              <xs:element name="value_description_add" type="valueDescriptionAddType"/>
              <xs:element name="value_description_delete" type="valueDescriptionDeleteType"/>
              <xs:element name="value_description_get_memory" type="valueDescriptionMemoryGetType"/>
              <xs:element name="value_description_report_memory"
                  type="valueDescriptionMemoryReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 125: Value Description XSD 2/2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="valueGetType">
    <xs:attribute name="value_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="valueReportType">
    <xs:attribute name="timestamp" type="xs:unsignedLong" use="required"/>
    <xs:attribute name="value_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="number" type="xs:double" use="optional"/>
    <xs:attribute name="string" type="xs:string" use="optional"/>
    <xs:attribute name="hexBinary" type="xs:hexBinary" use="optional"/>
  </xs:complexType>
  <xs:complexType name="valueSetType">
    <xs:attribute name="value_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="timestamp" type="xs:unsignedLong" use="required"/>
    <xs:attribute name="number" type="xs:double" use="optional"/>
    <xs:attribute name="string" type="xs:string" use="optional"/>
    <xs:attribute name="hexBinary" type="xs:hexBinary" use="optional"/>
  </xs:complexType>
  <xs:complexType name="valueGetLogType">
    <xs:attribute name="value_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="start_time" type="xs:unsignedLong" use="optional"/>
    <xs:attribute name="log_count" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="value_get" type="valueGetType"/>
              <xs:element name="value_report" type="valueReportType"/>
              <xs:element name="value_set" type="valueSetType"/>
              <xs:element name="value_get_log" type="valueGetLogType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 126: Value XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="infoType">
    <xs:attribute name="type_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="number" type="xs:unsignedLong" use="optional"/>
    <xs:attribute name="string" type="xs:string" use="optional"/>
    <xs:attribute name="hex" type="xs:hexBinary" use="optional"/>
  </xs:complexType>
  <xs:complexType name="partnerType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="info" type="infoType"/>
    </xs:sequence>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="groupPartnerType">
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="groupType">
    <xs:sequence>
      <xs:element name="partner" type="groupPartnerType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="partnerInformationGetType">
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="partnerInformationReportType">
    <xs:choice>
      <xs:sequence>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="partner" type="partnerType"/>
          <xs:element name="group" type="groupType"/>
        </xs:choice>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="partnerInformationSetType">
    <xs:choice>
      <xs:sequence>
        <xs:choice minOccurs="1" maxOccurs="unbounded">
          <xs:element name="partner" type="partnerType"/>
          <xs:element name="group" type="groupType"/>
        </xs:choice>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="partnerInformationDeleteType">
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="partnerInformationMemoryGetType">
  </xs:complexType>
  <xs:complexType name="partnerInformationMemoryReportType">
```

XML 127: Partner Information XSD 1/2

```
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="partner_information_get" type="partnerInformationGetType"/>
              <xs:element name="partner_information_report" type="partnerInformationReportType"/>
              <xs:element name="partner_information_set" type="partnerInformationSetType"/>
              <xs:element name="partner_information_delete" type="partnerInformationDeleteType"/>
              <xs:element name="partner_information_get_memory"
                  type="partnerInformationMemoryGetType"/>
              <xs:element name="partner_information_report_memory"
                  type="partnerInformationMemoryReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 128: Partner Information XSD 2/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="getType">
    <xs:attribute name="value_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="transport_mode" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="setType">
    <xs:attribute name="value_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="number" type="xs:double" use="optional"/>
    <xs:attribute name="string" type="xs:string" use="optional"/>
    <xs:attribute name="hexBinary" type="xs:hexBinary" use="optional"/>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="transport_mode" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="reportType">
    <xs:attribute name="my_value_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="transport_mode" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="invokeType">
    <xs:attribute name="action_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="timerType">
    <xs:attribute name="timer_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="actionType">
    <xs:choice minOccurs="1" maxOccurs="unbounded">
      <xs:element name="get" type="getType"/>
      <xs:element name="set" type="setType"/>
      <xs:element name="report" type="reportType"/>
      <xs:element name="invoke" type="invokeType"/>
      <xs:element name="timer_start" type="timerType"/>
      <xs:element name="timer_stop" type="timerType"/>
    </xs:choice>
    <xs:attribute name="action_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="actionGetType">
    <xs:attribute name="action_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="actionReportType">
    <xs:sequence>
      <xs:element name="action" type="actionType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="actionSetType">
    <xs:sequence>
      <xs:element name="action" type="actionType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="actionDeleteType">
    <xs:attribute name="action_id" type="xs:unsignedInt" use="optional"/>
```

XML 129: Action XSD 1/2

```
  <xs:complexType name="actionInvokeType">
    <xs:attribute name="action_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="actionMemoryGetType">
  </xs:complexType>
  <xs:complexType name="actionMemoryReportType">
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="action_get" type="actionGetType"/>
              <xs:element name="action_report" type="actionReportType"/>
              <xs:element name="action_set" type="actionSetType"/>
              <xs:element name="action_delete" type="actionDeleteType"/>
              <xs:element name="action_invoke" type="actionInvokeType"/>
              <xs:element name="action_get_memory" type="actionMemoryGetType"/>
              <xs:element name="action_report_memory" type="actionMemoryReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 130: Action XSD 2/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="calculationType">
    <xs:sequence>
      <xs:element name="left" type="calSubType" minOccurs="1" maxOccurs="1"/>
      <xs:element name="right" type="calSubType" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="method_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="calSubType">
    <xs:attribute name="value_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="partner_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="statemachine_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="timer_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="calendar_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="is_updated" type="xs:unsignedByte" use="optional"/>
    <xs:attribute name="constant_number" type="xs:double" use="optional"/>
    <xs:attribute name="constant_string" type="xs:string" use="optional"/>
    <xs:attribute name="constant_hexBinary" type="xs:hexBinary" use="optional"/>
  </xs:complexType>
  <xs:complexType name="calculationGetType">
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="calculationReportType">
    <xs:sequence>
      <xs:element name="calculation" type="calculationType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="calculationSetType">
    <xs:sequence>
      <xs:element name="calculation" type="calculationType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="calculationDeleteType">
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="calculationMemoryGetType">
  </xs:complexType>
  <xs:complexType name="calculationMemoryReportType">
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="calculation_get" type="calculationGetType"/>
```

XML 131: Calculation XSD 1/2

```
            <xs:element name="calculation_set" type="calculationSetType"/>
            <xs:element name="calculation_delete" type="calculationDeleteType"/>
            <xs:element name="calculation_get_memory" type="calculationMemoryGetType"/>
            <xs:element name="calculation_report_memory" type="calculationMemoryReportType"/>
          </xs:choice>
          <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
          <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
          <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

XML 132: Calculation XSD 2/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="executeType">
    <xs:attribute name="timer_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="after" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="action_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="timerGetType">
    <xs:attribute name="timer_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="timerReportType">
    <xs:sequence>
      <xs:element name="execute" type="executeType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="timerSetType">
    <xs:sequence>
      <xs:element name="execute" type="executeType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="timerDeleteType">
    <xs:attribute name="timer_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="timerMemoryGetType">
  </xs:complexType>
  <xs:complexType name="timerMemoryReportType">
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="timer_get" type="timerGetType"/>
```

XML 133: Timer XSD 1/2

```
              <xs:element name="timer_set" type="timerSetType"/>
              <xs:element name="timer_delete" type="timerDeleteType"/>
              <xs:element name="timer_get_memory" type="timerMemoryGetType"/>
              <xs:element name="timer_report_memory" type="timerMemoryReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 134: Timer XSD 2/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="taskType">
    <xs:attribute name="task_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="start" type="xs:unsignedLong" use="required"/>
    <xs:attribute name="action_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="end" type="xs:unsignedLong" use="optional"/>
    <xs:attribute name="repeat" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="weekdays" type="xs:unsignedByte" use="optional"/>
  </xs:complexType>
  <xs:complexType name="calendarGetType">
    <xs:attribute name="task_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="calendarReportType">
    <xs:sequence>
      <xs:element name="task" type="taskType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="calendarSetType">
    <xs:sequence>
      <xs:element name="task" type="taskType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="calendarDeleteType">
    <xs:attribute name="task_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="calendarTimezoneGetType">
  </xs:complexType>
  <xs:complexType name="calendarTimezoneSetType">
    <xs:attribute name="offset" type="xs:int" use="required"/>
  </xs:complexType>
  <xs:complexType name="calendarTimezoneReportType">
    <xs:attribute name="offset" type="xs:int" use="required"/>
  </xs:complexType>
  <xs:complexType name="calendarMemoryGetType">
  </xs:complexType>
  <xs:complexType name="calendarMemoryReportType">
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="calendar_get" type="calendarGetType"/>
              <xs:element name="calendar_report" type="calendarReportType"/>
              <xs:element name="calendar_set" type="calendarSetType"/>
```

XML 135: Calendar XSD 1/2

```
            <xs:element name="calendar_get_timezone" type="calendarTimezoneGetType"/>
            <xs:element name="calendar_set_timezone" type="calendarTimezoneSetType"/>
            <xs:element name="calendar_report_timezone" type="calendarTimezoneReportType"/>
            <xs:element name="calendar_get_memory" type="calendarMemoryGetType"/>
            <xs:element name="calendar_report_memory" type="calendarMemoryReportType"/>
          </xs:choice>
          <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
          <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
          <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

XML 136: Calendar XSD 2/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="transactionType">
    <xs:attribute name="calculation_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="action_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="goto_state_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="stateType">
    <xs:sequence>
      <xs:element name="transaction" type="transactionType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="state_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="statemachineGetType">
    <xs:attribute name="statemachine_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="state_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="statemachineType">
    <xs:choice minOccurs="1" maxOccurs="unbounded">
      <xs:element name="state" type="stateType"/>
    </xs:choice>
    <xs:attribute name="statemachine_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="statemachineReportType">
    <xs:sequence>
      <xs:element name="statemachine" type="statemachineType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="statemachineSetType">
    <xs:sequence>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="statemachine" type="statemachineType" minOccurs="1" maxOccurs="unbounded"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="statemachineDeleteType">
    <xs:attribute name="statemachine_id" type="xs:unsignedInt" use="optional"/>
    <xs:attribute name="state_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="statemachineStateType">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedInt">
        <xs:attribute name="statemachine_id" type="xs:unsignedInt" use="required"/>
      </xs:extension>
    </xs:simpleContent>
```

XML 137: State Machine XSD 1/2

```
  <xs:complexType name="statemachineGetStateType">
    <xs:attribute name="statemachine_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="statemachineSetStateType">
    <xs:sequence>
      <xs:element name="statemachine_state" type="statemachineStateType" minOccurs="1"
    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="statemachineReportStateType">
    <xs:sequence>
      <xs:element name="statemachine_state" type="statemachineStateType" minOccurs="0"
    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="statemachineMemoryGetType">
  </xs:complexType>
  <xs:complexType name="statemachineMemoryReportType">
    <xs:attribute name="count" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="free_count" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="statemachine_get" type="statemachineGetType"/>
              <xs:element name="statemachine_report" type="statemachineReportType"/>
              <xs:element name="statemachine_set" type="statemachineSetType"/>
              <xs:element name="statemachine_delete" type="statemachineDeleteType"/>
              <xs:element name="statemachine_get_state" type="statemachineGetStateType"/>
              <xs:element name="statemachine_report_state" type="statemachineReportStateType"/>
              <xs:element name="statemachine_set_state" type="statemachineSetStateType"/>
              <xs:element name="statemachine_get_memory" type="statemachineMemoryGetType"/>
              <xs:element name="statemachine_report_memory" type="statemachineMemoryReportType"/>
              <xs:element name="statemachine_get_state_memory" type="statemachineMemoryGetType"/>
              <xs:element name="statemachine_report_state_memory"
                  type="statemachineMemoryReportType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 138: State Machine XSD 2/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="firmwareInitType">
    <xs:attribute name="size" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="checksum" type="xs:hexBinary" use="required"/>
    <xs:attribute name="firmware_id" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="firmwareDataType">
    <xs:sequence>
      <xs:element name="chunk" type="xs:hexBinary" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="offset" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="firmwareUpdateStartType">
  </xs:complexType>
  <xs:complexType name="firmwareReportType">
    <xs:attribute name="status" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="expected_offset" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
  <xs:complexType name="firmwareInformationGetType">
  </xs:complexType>
  <xs:complexType name="firmwareInformationReportType">
    <xs:attribute name="size" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="firmware_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="received_size" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="chunk_size" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
```

XML 139: Firmware Update XSD 1/2

```
            <xs:element name="firmware_init" type="firmwareInitType"/>
            <xs:element name="firmware_data" type="firmwareDataType"/>
            <xs:element name="firmware_update_start" type="firmwareUpdateStartType"/>
            <xs:element name="firmware_report" type="firmwareReportType"/>
            <xs:element name="firmware_information_get" type="firmwareInformationGetType"/>
            <xs:element name="firmware_information_report" type="firmwareInformationReportType"/>
          </xs:choice>
          <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
          <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
          <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XML 140: Firmware Update XSD 2/2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="configStatusGetType">
  </xs:complexType>
  <xs:complexType name="configStatusReportType">
    <xs:attribute name="status" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="configModeSetType">
    <xs:attribute name="mode" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="config_status_report" type="configStatusReportType"/>
              <xs:element name="config_status_get" type="configStatusGetType"/>
              <xs:element name="config_mode_set" type="configModeSetType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 141: Configuration XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="statusReportType">
    <xs:attribute name="type_id" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="code" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="level" type="xs:unsignedInt" use="required"/>
    <xs:attribute name="data" type="xs:hexBinary" use="optional"/>
  </xs:complexType>
  <xs:complexType name="statusGetLevelType">
  </xs:complexType>
  <xs:complexType name="statusSetLevelType">
    <xs:attribute name="level" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <xs:complexType name="statusReportLevelType">
    <xs:attribute name="level" type="xs:unsignedInt" use="required"/>
  </xs:complexType>
  <!-- Main Element -->
  <xs:element name="network">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="status_report" type="statusReportType"/>
              <xs:element name="status_get_level" type="statusGetLevelType"/>
              <xs:element name="status_set_level" type="statusSetLevelType"/>
              <xs:element name="status_report_level" type="statusReportLevelType"/>
            </xs:choice>
            <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="device_id" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="go_to_sleep" type="xs:unsignedInt" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:unsignedInt" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 142: Status XSD

List of Tables

List of XMLs

List of Figures

List of Corrections