Lemonbeat
smart Device Language

Specification

Version 1.13
10/08/2017

lemonbeat

Contents

Chapter 1

Introduction

The Lemonbeat Protocol Stack is a protocol stack aimed to provide a solution for low-cost and low-power connectivity for devices that use batteries as a power source. The Lemonbeat Protocol Stack aims to solve the problems that exist in currently available protocols.
These problems are among others:

- Missing or bad security
- Bad robustness
- Interference with other technology
- Unmet requirements for a gateway centric system
- Duty cycle limits
- Limited range of the radio signals
- Bad routing strategies
- Not standards-based
- Suboptimal application layer

All layers and services in the Lemonbeat Protocol Stack from the Data Link layer up are based on Internet standards. The Lemonbeat smart Device Language (LsDL, also Lemonbeat Application layer) described in this document is self-descriptive, extendable and based on standard technology that is widely used on the Internet today.

1.1 Document revision history

Ver.	Rev.	Description	Authors
1	0	Initial draft version.	Andreas Madsen, Daniel Lux, Henrik Sorensen, Leni Lausdahl, Morten Frederiksen
1	3	Updated value types and units. Updated network inclusion. Removed unused tags and attributes.	Andreas Madsen, Daniel Lux, Henrik Sorensen, Morten Frederiksen
1	4	Changed memory IDs. Renamed conditions to calculation.	Andreas Madsen, Daniel Lux, Henrik Sorensen, Morten Frederiksen
1	5	Changed PHY header to better support forward error correction.	Andreas Madsen, Daniel Lux, Henrik Sorensen, Morten Frederiksen
1	6	Converted document to LaTeX. Change PHY header for Forward Error Correction. Updated services due to new XSDs. Added Status and Configuration Service. Removed Partner Link Service.	Andreas Bomholtz, Daniel Lux, Henrik Sorensen, Morten Frederiksen
1	7	Updated the Status Service.	Andreas Bomholtz, Daniel Lux, Henrik Sorensen, Morten Frederiksen
1	8	Added new generic MAC options. Added new special virtual values.	Andreas Bomholtz, Daniel Lux, Henrik Sorensen, Morten Frederiksen
1	9	Reviewed different radio modes, added explanations. Deleted description of logging for the Value Service. Added info how the value timestamp is handled.	Georg Werner
1	10	Deleted NTP/SGTIN details, updated spelling.	Georg Werner
1	11	Minor formal corrections.	Timo Dammes
1	12	Linguistic and formal review.	Beate Hoffmann
1	13	Layout plus terminology and content-related changes. Added missing device description types. Corrected wrong values in the configuration mode table.	Beate Hoffmann, Anton Abaschin, Christian Taedcke

Chapter 2

Lemonbeat smart Device Language

This chapter contains the following:

- A description of the Lemonbeat smart Device Language (LsDL)
- XML examples for the protocol (section 2.2)
- User stories for some scenarios and related XML solutions (section 2.3)

2.1 LsDL Description

The Lemonbeat smart Device Language is defined in XML to make messages human-readable as well as protocol messages easy to define. Since XML is too big to use in an embedded Application Protocol, 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 EXI specification is defined by the W3C (`www.w3.org`) and can be found at `http://www.w3.org/TR/2011/REC-exi-20110310/`. This way an example XML document of 274 bytes can be compressed to 4 bytes without loss of information. See XML 2.1 and XML 2.2 for the compression of an XML message of 1253 bytes into an EXI message of 83 bytes.

2.1.1 Service types and related ports

The Lemonbeat Software Stack contains predefined Lemonbeat Services (for a description of the different Lemonbeat Services see Section 2.1.4).

The service type used can be determined by the destination port of the message. The services are reachable via TCP and UDP. The ports are defined in Table 2.1. Each message contains information about the source and destination port. The destination port is the defined service port. The source port can be a randomly selected port from 20128 to 20256 or one of the defined service ports. All answers are sent back to the source port, so the sender knows that the incoming message is the answer. So when sending a `value_get` message, the destination port will be the value port and the source port will be a random port. The reply will be sent back with the ports swapped, so that the source port is the value port.
Note: Currently only UDP is supported for devices that use radio-type Wake-On-Radio or event listener.
The port that the devices listen on starts from port 20000. The ports can be compressed, so port 20000 is mapped to port 0 and port 20256 to port 256 as defined in Table 2.1.

```
<?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 as 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 as EXI version (values in hexadecimal notation)

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 2.1: List of services and port numbers

2.1.2 Network management

To include a device into a network the following steps are required:

Device A starts up and sends a Device Description message, signaling that it wants to be included into the network. The network controller receives the Device Description message and decides if device A should be included or not.

If the network controller decides to include device A, the controller 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 generates 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 is shown in Table 2.2.

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 2.2: The format of the inclusion message

Afterwards this message is 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.

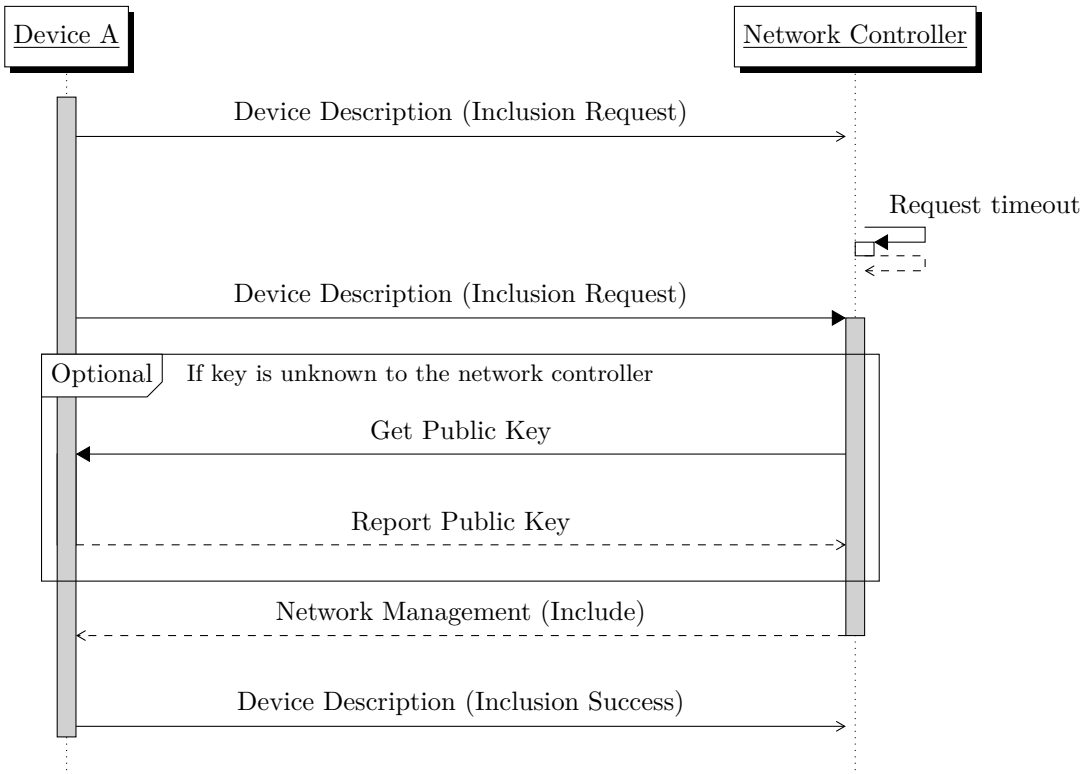

Figure 2.1: Inclusion

In addition to the network key, there is a number that informs the device about the number of bytes it should use from its MAC address as its new address. For example, the new address will be converted from `0x1234567890ab` to `0x90ab` if the address size is 2.

When the device 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 addresses it listens on. Device A is now included in the network and can communicate with 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 performs a 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 the exclusion of a device.

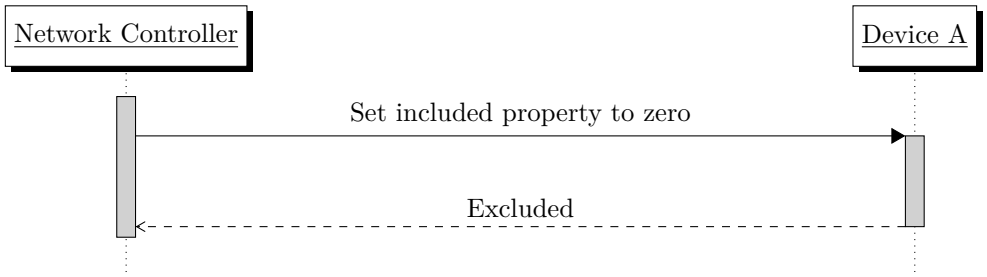

Figure 2.2: Exclusion

2.1.3 NTP

The Network Time Protocol is used to synchronize the clocks of Lemonbeat devices. NTP uses the UDP on port number 123. Lemonbeat devices must implement the NTP protocol as defined in RFC-5905.

2.1.4 Device configuration

Normally the behavior of a device is hard-coded in the application of the device. Sometimes the device supports a few 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 attained by using state machines. These state machines can be constructed and tested on a PC and then 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 supports configuration, the device can be configured. Every time a device is configured, the new configuration has to be saved before it becomes active. The Configuration Service is used to store and reset the device configuration. The Configuration Service is described in Section 2.1.4.8.

2.1.4.1 Virtual values

A device can support virtual values. Virtual values are values that can be configured externally. These values can be used as a variable when performing complex calculations and state machines. If a virtual value is added with a type that the device does not support, it will sent a status report back.
There are specific types of virtual values that results in a pre-defined behavior. These types and the resulting behaviors are described in Table 2.3. All of these special virtual values must be configured with the `persistent` property set to 0.

Name	Type	Mode	Unit	Description
Stop Watch	TIME	R/W	ms	The value can be set to 0 to reset the stop watch. Whenever 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 2.3: Special virtual values

2.1.4.2 Partner Service

The Partner Service is used to map an address to a simple ID. This way other devices can be adressed by this simple ID. This partner ID can be used in all the other services to address other devices. A partner information contains an address and information about how to communicate with the device. Only devices which are configured as partners can talk to the device. If a multicast address is configured as a partner, the device will listen and accept messages sent to this multicast address. The Partner Service also supports grouping multiple partners into a group with an ID. If the device sends a message to a group, it will be sent to all the partners in the group. If one or more of the partners in the group is a multicast address, the message is sent first to all multicast addresses in the group.

2.1.4.3 Timer Service

The Timer Service is used for executing actions with a delay. The delay is specified in milliseconds. Furthermore, conditions can be connected with a timer, so that some conditions 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 its states.

2.1.4.4 Calendar Service

Using the Calendar Service, a device can be configured to execute an action at a specific moment in time. This calendar task can be set to repeat at a specific interval. A Filter value enables the user to set the weekday on which the Calendar task should be executed.
The calendar will signal the state machine that the calendar task has triggered, so that the state machine can check its states.
The Calendar Service will only execute if the device has been synchronized with NTP.

2.1.4.5 Action Service

An action can get or set a value on the device itself or on another 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 current and the last action are similar to the same partner, the current action is combined with the last action instead of being added to the queue. See Figure 2.3 for the flow chart of the action enqueue process.

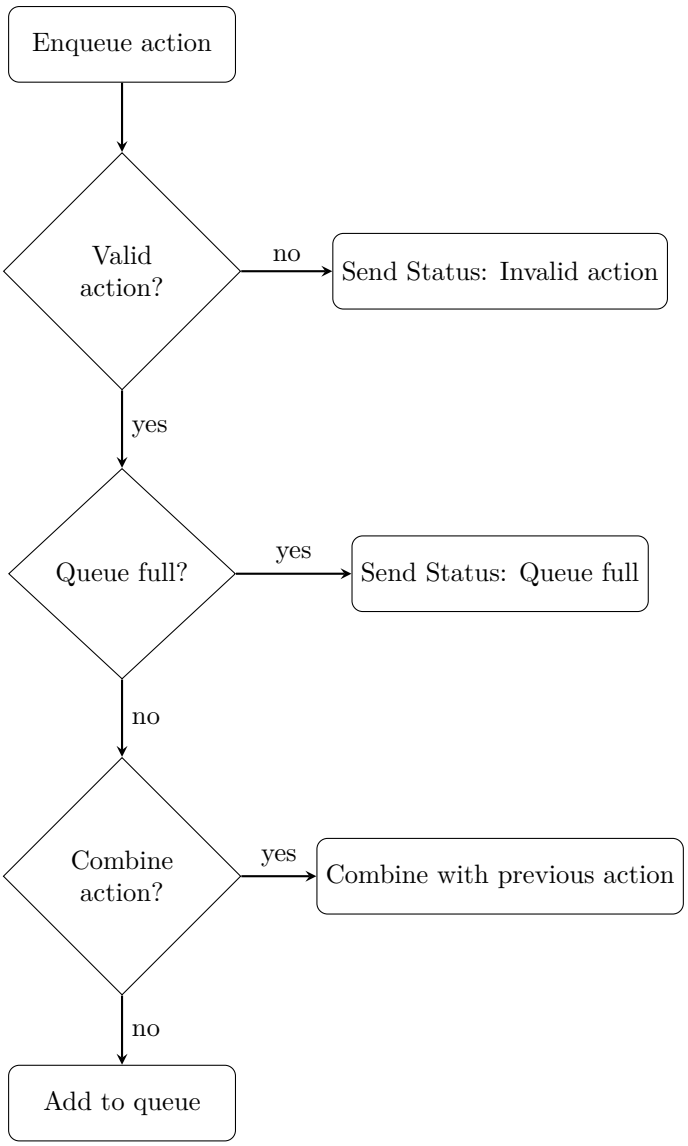

Figure 2.3: Action enqueue

2.1.4.6 Calculation Service

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

2.1.4.7 State Machine Service

A state machine consists of states and transactions. A state can be addressed by its ID. A transaction can have a calculation, an action and a state that the state machine should go to. If no calculation is attached to the transaction, it is interpreted as always true. If there is no next state that the transaction should move to, the state machine will keep its current state.
State machines can be triggered by 4 different events: timer event, calendar event, incoming value report and local value update. When the State Machine Service is triggered, it will loop through all the state machines and check the current state of each machine. It will execute the first transaction that is true from the current state of the machine. It will only execute each transaction once per event. This way there will be no runaway loops due to invalid configuration of the state machine. When the event is handled by the state machine, 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 through all the state machines again. For the flow chart of state machine execution see Figure 2.4 on the next page.

2.1.4.8 Configuration Service

The Configuration Service is used to persist and to apply the current configuration. When the device receives a new configuration, the Configuration status is marked as started and the device sends a Status message. This message contains the information that the status has changed to started. When the configuration is started, state machine, timer and calendar are halted, until the configuration is switched back to idle again. This can be done by an incoming configuration mode that set the mode or by a timeout that occurs if there has been no new configuration for a specific time. If a timeout occurs, the device will roll back any received configurations.

2.1.4.9 Status Service

The Status Service is used to report errors in the device. A status report contains the error type (type ID) and an error code that describes what happened. There is also an optional data field that can be used to report additional information about the error.
The application can send error messages using the Status Service. The application has its own type ID, but the error code is application-specific.

2.1.4.10 Firmware Update Service

The Firmware Update Service is used to update boot loader, stack and application of a device. The application can be updated without updating the boot loader and stack. Thus, changes to the application code can easily be updated on a device, because the application code is limited in size. The boot loader can only be updated together with the stack and application to ensure compatibility between boot loader, stack and application.

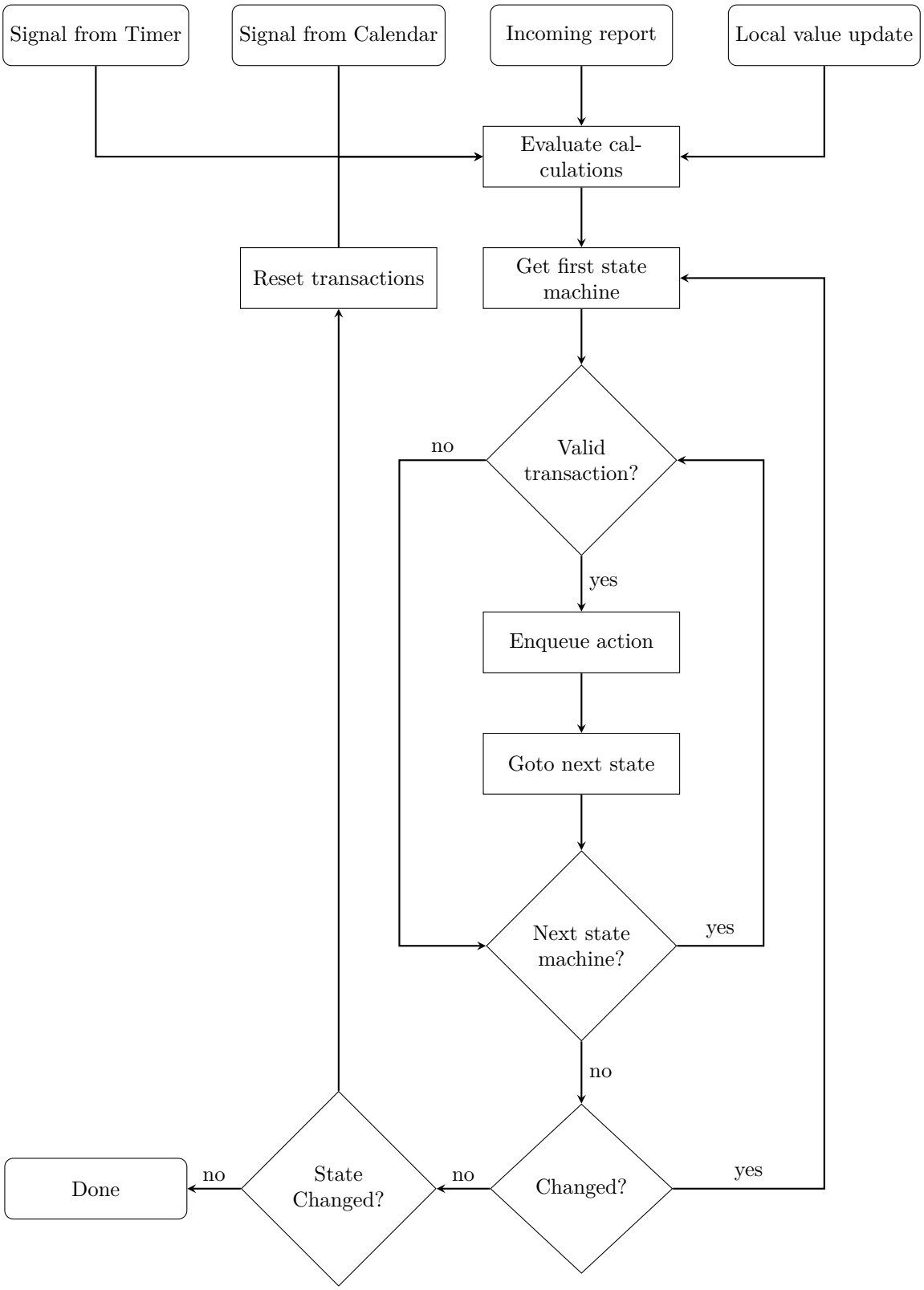

Figure 2.4: State machine executing

2.2 XML description

This section describes the XML messages. In every subsection there are descriptions of XML tags and examples of how to use them.

2.2.1 Network and device

The `network` and `device` tags are used to encapsulate all messages. They both have an attribute labeled version. The version fields in the `network` tag and `device` tag describe which version of the message format is used for the respective nodes. The `device` tag also has a `device_ID` attribute for addressing multiple devices in a single message and a `go_to_sleep` attribute for defining the time span before the device should go to sleep.

2.2.1.1 Tag description

The message tag for the network and device is described in Table 2.4 on the next page.

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 is present, the message is for the main device.
	`go_to_sleep`	No	Time to wait in milliseconds before the device goes to sleep.

Table 2.4: Network and device tags

2.2.1.2 Example

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

2.2.1.3 Network and device tags

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

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

XML 2.3: Encapsulation using network and device tags

2.2.2 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.

2.2.2.1 Tag description

The message tag for the network management is described in Table 2.5.

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

Table 2.5: Network management tags

2.2.2.2 Inclusion data

The inclusion data is RSA-encrypted with the public key of the device. The decrypted data consists of an AES controller key, the AES-encrypted network key with the controller key, and a CRC of the complete message. The format is described above in Table 2.2.

2.2.2.3 Examples

The following section contains an XML example for using the `network_include` messages.

Network include
To include a device a `network_include` message with a controller key and the network key must be sent from the network controller to the device. See XML 2.4 for an example of this message and Table 2.6 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

Type	Data
Controller key (hex)	0102030405060708090A0B0C0D0E0F00
Network key (hex)	00112233445566778899AABBCCDDEEFF
Encrypted Network key (hex)	E9855F797E25CDF8EC9A4ECB83E6632B
CRC (hex)	5948
Complete message (hex)	0102030405060708090A0B0C0D0E0F00E9 855F797E25CDF8EC9A4ECB83E6632B5948
RSA public key (decimal)	65537
RSA common key (decimal)	54470715462426500829311500310525272711904016356297127814065552874241070617335139097638779009662600671942632638047851873648163272789081193108015117548459
RSA encrypted message (hex)	014CAE536800D9114DB19EEEC7EF857658BBA06F145B00B09ACE371F003BB9C2FBF2C500768DEEB22FCAC92DEC605CB998C244EBAE48AD8B10CEE139209796F1

Table 2.6: The values used in the example

2.2.3 Public Key

The Public Key message is used to get and report the public key of the device.

2.2.3.1 Tag description

The message tag for the public key is described in Table 2.7.

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 2.8 for allowed values.

Table 2.7: Public key tags

2.2.3.1.1 Key types

The supported public key types are listed in Table 2.8.

Key Type ID	Key Type
1	RSA
...	...

Table 2.8: Public key types

2.2.3.2 Examples

The following section contains `publickey_get` and `publickey_report` XML message examples.

Get a public key

To get a public key, a `publickey_get` message should be sent to the device. See XML 2.5 for an example of this 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

To report a public key, a `publickey_report` message should be sent from the device. See XML 2.6 for an example of this 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

2.2.4 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, This tag indicates whether the service allows, for example, memory information, device description or value description. Furthermore a `version` tag indicates the highest supported version of this service.

2.2.4.1 Tag description

The message tag for the service description is described in Table 2.9.

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

Table 2.9: Service description tags

The child tags for the `service_description_report` message are described in Table 2.10.

XML tag	Attribute	Required	Description
`service`	`--`	Yes	The services the device supports.
	`service_id`	Yes	The service ID. See Table 2.11.
	`version`	Yes	The maximum supported version of the service.

Table 2.10: Service description report message child tags

2.2.4.1.1 Service

The valid options for the `service_id` attributes in the service are listed in Table 2.11.

Service ID	Device 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	Channel scan
14	Status
15	Configuration
...	...

Table 2.11: Service description types

2.2.4.2 Example

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

Get a service description
To get a service description, a `service_description_get` message should be sent to the device. See XML 2.7 for an example of this 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 this 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

2.2.5 Memory Information

The Memory Information service provides information about the current and the maximum number of timers, actions, etc. for a specific device.

2.2.5.1 Tag description

The message tag for the memory information is described in Table 2.12.

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

Table 2.12: Memory information tags

The child tags for the `memory_information_report` message are described in Table 2.13.

XML tag	Attribute	Required	Description
`memory_information`	`--`	Yes	The memory that the device supports.
	`memory_id`	Yes	The service ID, see Table 2.14.
	`count`	Yes	The maximum number of the services that the device supports.
	`free_count`	Yes	The number of free memory slots.

Table 2.13: Memory_information_report child tags

2.2.5.1.1 Memory

The valid options for the `memory_id` attributes in the `memory_information` tag are listed in Table 2.14.

Memory ID	Description
1	Value
2	Partner Information
3	Action Items
4	Calculation
5	Timer
6	Calendar
7	Statemachine
8	Statemachine Transactions
...	...

Table 2.14: Memory IDs

2.2.5.2 Examples

The following section contains XML examples for using the `memory_information` message.

Get the memory information
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 this 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 this 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

2.2.6 Device Description

The Device Description message is used to describe information about the device, e.g. type, manufacturer, etc. It is maintained using a report, get and set message.

2.2.6.1 Tag description

The message tag for the device description is described in Table 2.15.

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 2.15: Device Description tags

The child tags for the `device_description_report` and `device_description_set` message are described in Table 2.16.

XML tag	Attribute	Required	Description
`info`	`--`	Yes	Set a device property.
	`type_id`	Yes	The ID of the information. See Table 2.17.
	`number`	No	A number value.
	`string`	No	A string value.
	`hex`	No	A hexBinary value.

Table 2.16: Device Description child tags

2.2.6.2 Device description types

The device description types are described in Table 2.17. Settable types are formatted in bold.

ID	Name	Type	Description
1	Type	Number	Manufacturer type ID.
2	Manufacturer	Number	The manufacturer of the device. See Table 2.20 for a list of manufacturers.
3	SGTIN	Hex	Serialized Global Trade Item Number, 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 boot loader.
7	Stack Version	String	The version of the stack.
8	Application Version	String	The version of the application.
9	Protocol	Number	The communication protocol that the device uses. See Table 2.18 for a list of protocols.
10	Product	Number	Manufacturer 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 2.19.
14	**Wakeup Interval**	Number	The time in milliseconds that the device is offline between 2 consecutive RX-active periods. Currently set to 333 ms.
15	Wakeup Offset	Number	The offset in milliseconds needed for the calculation of the RX-active period. *Not needed in the current version.*
16	**Wakeup Channel**	Number	The radio channel that the device listens on for wake-up frames.
17	**Channel Map**	Hex	The current channel map. The channel map will always have 4 channels defined. 2 of them will be the synchronization channels.
18	Channel Scan Time	Number	The time is takes to scan a single channel.
19	IPv6 Address	Hex	IPv6 address.
20	**Wakeup Now**	Number	The time the device should stay awake. Only valid in the Partner Service.
21	Diversity Mode	Number	The diversity mode used in the device: 0 = off, 1 = on.
22	Reserved		
23	Reserved		
24	Reserved		
25	Reserved		
26	Reserved		
27	Reserved		
28	Reserved		
29	Reserved		
30	Reserved		
31	Reserved		
32	Reserved		

Table 2.17: Device description types. Types in **bold** are settable

2.2.6.2.1 Protocol

The different protocols are described in Table 2.18.

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

Table 2.18: Protocols

2.2.6.2.2 Radio Mode

The Radio Mode describes how a device communicates over the air. The default value for this is "Always Online". Therefore, this information can be omitted in a `device_description_report` tag if the device uses the default Radio Mode. The different Radio Modes are defined in Table 2.19.

Radio Mode ID	Description
0	Always Online
1	Wake-on-Radio (only UDP is supported)
2	Wake-on-Event (only UDP is supported)
3	TX only
4	RX only

Table 2.19: Radio Mode

2.2.6.2.3 Manufacturer

The different manufacturers are defined in Table 2.20.

Manufacturer ID	Manufacturer
1	RWE
2	Seluxit
3	...
4	Lemonbeat
...	...

Table 2.20: Manufactures

2.2.6.2.4 SGTIN

Every device must have a unique identification number that is assigned to the device during the manufacturing process. The identification number of the device must remain the same throughout the entire life cycle of the device. The identification number used is a 96 bits Serialized Global Trade Item Number (SGTIN-96), which is a standard for creating identification numbers.
SGTIN-96 is specified in the "EPC global Tag Data Standards Version 1.4"[1].

2.2.6.2.5 Channel map

The channel map is a bit map of the selected channels that the device uses for communication. The first bit (LSB) is channel 1 and the last bit (MSB) is channel 32.

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

2.2.6.3 Examples

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

Get a device description
To get a device description, a `device_description_get` message must be sent to the device. See XML 2.11 for an example of this 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 a device description

Report a device description
To report a device description, the `device_description_report` message plus related attributes must be sent from the device as described earlier. For an example of this message see XML 2.12.

```
<?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 a device description

Set device description properties
To set a device description property, the `device_description_set` message plus related attributes must be sent to the device with as described earlier. For an example of this message see XML 2.13.

```
<?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 a device description property

2.2.7 Value Description

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

A number value has the attributes `min`, `max` and `step`. If a value is outside the min-max range, it is adjusted so that it is equal to the minimum or maximum rage respectively. If the value does not adhere to the step size, it is rounded.

2.2.7.1 Tag description

The message tag for the value description is described in Table 2.21.

XML tag	Attribute	Required	Description
value_description_get	--	Yes	Get the description value.
	value_description_get	No	ID of the description value. To get all 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	Deletes 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 unused number of value descriptions the device supports.

Table 2.21: Value description tags

The child tags for value_description_report and value_description_add are described in Table 2.22.

XML tag	Attribute	Required	Description
value_description	--	Yes	The value description.
	value_id	Yes	ID of the description value.
	type_id	Yes	The value type, see Table 2.25.
	mode	Yes	Possible interaction with value: *Read Only* (from the device) *Read/Write* (to and from the device) *Write Only* (to the device)
	persistent	Yes	Specifies if the value is persisted during a power cycling: 0 = Not persistent 1 = Is persistent
	name	No	The name of the device.
	min_log_interval	No	Minimum time between value logs in seconds.
	max_log_values	No	Maximum numbers of logged values for the device.
	virtual	No	Specifies if the value is a virtual value.

Table 2.22: value_description_report and value_description_add message child tags

The child tags for `value_description` are described in Table 2.23.

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

Table 2.23: Value description child tags

The child tags for `string_format` are described in Table 2.24.

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

Table 2.24: Value description string_format child tags

2.2.7.1.1 Type

The valid options for the `type_id` attribute in the Value Description are listed in Table 2.25. The `type_id` attribute is required and should be set to a valid option.

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 2.25: Value description types

2.2.7.1.2 Unit

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

2.2.7.2 Examples

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

Get a specific value description

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 this 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

To get all value descriptions, 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 this 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 tags `value_description` and `number_format`. See XML 2.16 for an example of this 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 this 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 this 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 2.19 for an example of this 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_delete value_description_id="5" />
  </device>
</network>
```

XML 2.19: Delete virtual value description

2.2.8 Value

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

If the device is not synchronized with NTP, the reported timestamp will be zero.
If the device is synchronized with NTP, the timestamp will contain a valid datetime.

2.2.8.1 Tag description

The message tag for the value is described in Table 2.26.

XML tag	Attribute	Required	Description
`value_get`	`--`	Yes	The `value_get` tag is used to get all 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 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 indicates when the current value was sent.
`value_set`	`--`	Yes	The `value_set` tag is used to set multiple 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 indicating when the current value was sent. Unix-timestamp in milliseconds.

Table 2.26: Value tags

2.2.8.2 Examples

This section shows XML examples of how to use the `value_get`, `value_report`, `value_report`, `value_set` and `value_set` messages.

Get a specific value
To get a specific value, a `value_get` message has to be sent to the device with a value ID. See XML 2.20 for an example of this 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 2.20: Get the value with ID 1

Get all values
To get all values, a `value_get` message has to be sent to the device without a value ID. See XML 2.21 for an example of this 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 2.21: Get all values

Report a specific number value
To report a specific number value, the `value_report` message has to be sent from the device with a value ID and a number. See XML 2.22 for an example of this 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 2.22: 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 device with a value ID and a string. See XML 2.23 for an example of this 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 2.23: 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 2.24 for an example of this 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 2.24: Report of all the values

Set a specific number value
To set a specific number value, a `value_set` message has to be sent with a value ID and a number. See XML 2.25 for an example of this 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 2.25: Set the value with ID 1 to 55

Set a specific string value
To set a specific string value, a `value_set` message has to be sent with a value ID and a string. See XML 2.26 for an example of this 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 2.26: Set the value with ID 2 to ON

Set multiple values

To set multiple values, a `value_set` with a value ID and a number or a string. See XML 2.27 for an example of this 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 2.27: Set the value with ID 1 to 55 and the value with ID 2 to ON

2.2.9 Partner Information

The Partner Information message is used to obtain detailed information about existing partners and groups. Furthermore, it is used to create additional information or to delete information from partners and groups.

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 a group.

A Partner is described by an address, an encryption key and a radio mode.
A group is described as a list of partners for the purpose of sending messages to all partners in the group at once, by using a multicast address. Adding a group to another group is not allowed.

2.2.9.1 Tag description

The message tags for the partner information are described in Table 2.27.

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 unused number of partners the device supports.

Table 2.27: Partner tags

The child tags for the `partner_information_report` and `partner_information_set` message are described in Table 2.28.

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 together and to a multicast address. *If not defined, partner must be defined.
	`partner_id`	Yes	ID of the partner.

Table 2.28: Partner information child tags

The child tags for partner are described in Table 2.29.

XML tag	Attribute	Required	Description
`info`	`--`	Yes	Set a device property.
	`type_id`	Yes	The ID of the information. See Table 2.30.
	`number`	No	A number value.
	`string`	No	A string value.
	`hex`	No	A hexBinary value.

Table 2.29: Partner Information partner child tags

The valid values for `type_id` in partner information is described in Table 2.30.

ID	Name	Type	Description
13	Radio Mode	Number	The partners radio mode. See Table 2.19.
14	Wakeup Interval	Number	The offset in milliseconds indicating when the partner starts to wake up. **Currently set to fixed 333 ms**.
15	Wakeup Offset	Number	How often in milliseconds the partner will wake up. *Not in use right now. For future purposes*
16	Wakeup Channel	Number	The radio channel that the partner listens on to see if it has to wake up.
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 awake.

Table 2.30: Device Description Types. Types in **bold** are settable.

The child tags for the group are described in Table 2.31.

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

Table 2.31: Group child tags

2.2.9.2 Using partner information to control devices supporting sleep mode

Controlling sleeping devices (i.e., waking them up or keeping them alive when they are awake) is handled with the `partner_information` on the Dongle. You have to set up a partner in order to actively wake up a WoR device or to keep an event listener awake that just sends a message to the Dongle.

2.2.9.2.1 Handling Wake-on-Radio (WoR) devices

A Wake-on-Radio device is a cyclic listening device that can be woken up by the gateway using a burst. Currently the device is listening 3 times per second for one millisecond for incoming packets. The WoR device listening cycle is currently 333 ms. The awake-listening-time is around 1 ms. The WoR device can be woken up by the gateway using a wake-up burst. The wake-up process is handled as follows:

- Configure the target device that you want to wake up as a partner on the gateway.
- Set the channel (default = 3), the Radio Mode, and the time that the device should wake up (WakeUpNow in milliseconds).
- After setting the partner the gateway tries immediately to wake up the device.
- After a successful wake-up the WoR device answers with an Awake-Now-Status-Report to inform the gateway that it is awake now.
- The gateway itself can now control the awake-phase of the device by using the go_to_sleep attribute in the device tag of the LsDL/XML message.

See XML 2.28 for an example of this message for a WoR Partner.

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="urn:partner_informationxsd" version="1">
  <device device_id="1" version="1">
    <partner_information_set>
      <partner partner_id="2">
        <!-- Radio Mode -->
        <info number="1" type_id="13"></info>
        <!-- Wakeup Channel -->
        <info number="3" type_id="16"></info>
        <!-- Channel Map -->
        <info hex="10080804" type_id="17"></info>
        <!-- Channel Scan Time -->
        <info number="10000" type_id="18"></info>
    <!-- Wakeup now -->
    <info type_id="20" number="20000"></info>
        <!-- IPv6 Address -->
        <info hex="FC00000000000000000694BBAE001686" type_id="19"></info>
      </partner>
    </partner_information_set>
  </device>
</network>
```

XML 2.28: Wake up the partner with the ID 2

2.2.9.2.2 Handling Event-Listening devices

The event listener is only reachable for other devices by manual interaction (e.g., a pressed button) or by an event that fires because some internal operations have been scheduled. An example might be a temperature sensor that transmits its value each hour. It can not be woken up directly from the gateway. You have to wait until one of the above mentioned events occurs. Then the procedure of controlling the awake-phase is similar to the WoR listeners:

- Configure the target device that you want to wake up as a partner on the gateway. Set the channel (default = 3), the Radio Mode (for event listener = 2), and the time that the device should wake up (WakeUpNow in milliseconds).
- The gateway keeps this partner information until it receives a message from the event listener. After the gateway receives a message (e.g., a value report) from the event listener, it immediately sends a WakeUp command to the target to keep it awake.
- After receiving the wake-up burst, the device will send a status report to inform the gateway that it is awake (similar to the WoR listener).
- The gateway itself can now control the awake-phase of the device by using the `go_to_sleep` attribute in the `device` tag of the LsDL/XML message.

2.2.9.3 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

To get a specific partner, a `partner_information_get` message has to be sent to the device with a partner ID. See XML 2.29 for an example of this 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 2.29: Get the partner with the ID 1

Get all partners

To get all partners, a `partner_information_get` message has to be sent to the device without a partner ID. See XML 2.30 for an example of this 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 2.30: 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 2.31 for an example of this 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="fc000000000000000100000000000012"/>
      </partner>
    </partner_information_report>
  </device>
</network>
```

XML 2.31: 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 2.32 for an example of this 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="fc000000000000000100000000000012"/>
      </partner>
      <partner partner_id="2">
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc000000000000000100000000000017"/>
      </partner>
      <group partner_id="3">
        <partner partner_id="1"/>
        <partner partner_id="2"/>
      </group>
    </partner_information_report>
  </device>
</network>
```

XML 2.32: Report all partners

Set a specific partner

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 2.33 for an example of this 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="fc000000000000000100000000000012"/>
      </partner>
    </partner_information_set>
  </device>
</network>
```

XML 2.33: Set the partner with ID 1

Set multiple partners

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 2.34 for an example of this 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="fc000000000000000100000000000012"/>
      </partner>
      <partner partner_id="2">
        <!-- IPv6 Address -->
        <info type_id="19" hex="fc000000000000000100000000000017"/>
      </partner>
      <group partner_id="3">
        <partner partner_id="1"/>
        <partner partner_id="2"/>
      </group>
    </partner_information_set>
  </device>
</network>
```

XML 2.34: Set multiple partners

Delete a specific partner

To delete a specific partner, a `partner_information_delete` message has to be sent to the device with a partner ID. See XML 2.35 for an example of this 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 2.35: Delete the partners with ID 1

Delete all partners

To delete all partners, a `partner_information_delete` message has to be sent to the device without a partner ID. See XML 2.36 for an example of this 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 2.36: Delete all partners

2.2.10 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 utilizing a timer to determine the runtime of a specific action.

2.2.10.1 Tag description

The message tag for the action is described in Table 2.32.

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 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 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 unused number of actions the device supports.

Table 2.32: Action tags

The child tags for the `action_report` and `action_set` message are described in Table 2.33.

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

Table 2.33: Action_report and action_set child tags

The child tags for the action are described in Table 2.34 on the next page.

XML tag	Attribute	Required	Description
get	--	No	Get a value from the specified device.
	value_id	Yes	ID of the value.
	partner_id	No	ID for the device the value should be retrieved from. If no partner_id exists, the value is from the device itself.
	transport_mode	No	The transport mode the action should be sent with. See Table 2.35
set	--	No	Set a value on the specified device.
	value_id	Yes	ID of the value.
	partner_id	No	ID for the device on which the value should be set. If no partner_id exists, the value will be set on the device itself.
	number	Yes*	The number value. *If not defined, number, string 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 sent with. See Table 2.35.
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 the report should be sent to. If no cpartner_id exists, the report will be sent to the device itself.
	transport_mode	No	The transport mode the action should be sent with. See Table 2.35.
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 2.34: Action child tags

2.2.10.1.1 Transport mode

Mode	Name	Description
0	UDP	User Datagram Protocol. A fast and simple, but unreliable protocol.
1	TCP	Transmission Control Protocol. A reliable, but slow protocol.

Table 2.35: Transport mode

2.2.10.2 Examples

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

Get a specific action
To get a specific action, an `action_get` message has to be sent to the device with an action ID. See XML 2.37 for an example of this 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 2.37: Get the action with ID 1

Get all actions
To get all actions, an `action_get` message has to be sent to the device without an action ID. See XML 2.38 for an example of this 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 2.38: 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 2.39 for an example of this 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 2.39: 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 2.40 for an example of this 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 2.40: Report all actions

Set a specific action

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 2.41 for an example of this 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 2.41: Set the action with ID 1

Set multiple actions

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 2.42 for an example of this 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"/>
        <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 2.42: Set multiple actions

Delete a specific action

To delete a specific action, an `action_delete` message has to be sent to the device with an action ID. See XML 2.43 for an example of this 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_delete action_id="5"/>
  </device>
</network>
```

XML 2.43: Delete the action with ID 5

Delete all actions

To delete all actions, an `action_delete` message has to be sent to the device without an action ID. See XML 2.44 for an example of this 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_delete/>
  </device>
</network>
```

XML 2.44: Delete all actions

2.2.11 Calculation

The Calculation message is used to perform specific calculations.
The Calculation message is maintained by get, report, set and delete tags. The `method_id` attribute specifies the operator of the calculation, e.g. addition, subtraction, multiplication or division. The result of a calculation contains two operands, a left and a right, which can be read-out separately.

2.2.11.1 Tag description

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

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 unused number of calculations the device supports.

Table 2.36: Calculation tags

The child tags for the `calculation_report` and `calculation_set` message are described in Table 2.37.

XML tag	Attribute	Required	Description
`calculation`	`--`	Yes	The calculation tag is used to 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 2.38.

Table 2.37: Calculation message child tags

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

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, `calculation_id` must be defined.
	`calculation_id`	Yes*	ID of the calculation*. If no `calculation_id` is present, `value_id` must be defined.
	`partner_id`	No	ID for the device with the value. If no `partner_id` is present, 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. *Currently not supported in calculations.*
	`constant_number`	No	A constant number value.
	`constant_hexBinary`	No	A constant hex value. *Currently not supported in calculations.*
	`is_updated`	No	If this is set to the value 1, the calculation will return true if the value is updated.

Table 2.38: Calculation child tags

2.2.11.1.1 Method

The valid options for the `method_id` attribute in the calculation are listed in Table 2.39. The `method_id` attribute is 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 2.39: Method types

2.2.11.2 Examples

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

Get a specific calculation

To get a specific calculation, a `calculation_get` message has to be sent to the device with a calculation ID. See XML 2.45 for an example of this 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 2.45: Get the calculation with ID 1

Get all calculations

To get all calculations, a `calculation_get` message has to be sent to the device without a calculation ID. See XML 2.46 for an example of this 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 2.46: 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 2.47 for an example of this 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 2.47: 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 2.48 for an example of this 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 2.48: Report all calculations

Set a specific calculation

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 2.49 for an example of this 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 2.49: Set the calculation add-method

Set multiple calculations

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 2.50 for an example of this 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 2.50: Set multiple calculations

Delete a specific calculation

To delete a specific calculation, a `calculation_delete` message has to be sent to the device with a calculation ID. See XML 2.51 for an example of this 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 2.51: Delete the calculation with ID 1

Delete all calculations
To delete all calculations, a `calculation_delete` message has to be sent to the device without a calculation ID. See XML 2.52 for an example of this 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 2.52: Delete all calculations

2.2.12 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 perform a specified action after the 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`.

2.2.12.1 Tag description

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

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 unused number of timers the device supports.

Table 2.40: Timer tags

The child tags for the `timer_report` and `timer_set` message are described in Table 2.41.

XML tag	Attribute	Required	Description
`execute`	--	Yes	The timer will execute the action after the specified milliseconds in the `after` attribute.
	`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 no `action_id` is present, an event will be sent to the state machine.

Table 2.41: Timer message child tags

2.2.12.2 Examples

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

Get a specific timer
To get a specific timer, a `timer_get` message has to be sent to the device with a timer ID. See XML 2.53 for an example of this 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 2.53: Get the timer with ID 1

Get all timers
To get all timers, a `timer_get` message has to be sent to the device without a timer ID. See XML 2.54 for an example of this 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 2.54: 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 2.55 for an example of this 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 2.55: 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 2.56 for an example of this 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 2.56: 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 2.57 for an example of this 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 2.57: Report all timers

Set a specific timer

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 2.58 for an example of this 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 2.58: Set the timer with ID 2 to execute action 3 after 2000 ms

Set multiple timers

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 2.59 for an example of this 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="1" after="2000" action_id="2"/>
      <execute timer_id="2" after="4000"/>
    </timer_set>
  </device>
</network>
```

XML 2.59: Set the timers with ID 1 and 2

Delete a specific timer

To delete a specific timer, a `timer_delete` message has to be sent to the device with a timer ID. See XML 2.60 for an example of this 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_delete timer_id="1"/>
  </device>
</network>
```

XML 2.60: Delete the timer with ID 1

Delete all timers

To delete all timers, a `timer_delete` message has to be sent to the device without a timer ID. See XML 2.61 for an example of this 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_delete/>
  </device>
</network>
```

XML 2.61: Delete all the timers

2.2.13 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. It will 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 a specific number of seconds between each pass. The `weekdays` attribute specifies on which weekdays the repeated task should be executed.

2.2.13.1 Tag description

The message tag for the calendar is described in Table 2.42.

XML tag	Attribute	Required	Description
`calendar_get`	`--`	Yes	The `calendar_get` tag is used to get all calendar tasks. See examples below.
	`task_id`	No	ID of the task. To get all calendar tasks, omit `task_id`.
`calendar_report`	`--`	Yes	The `calendar_report` tag is used to report all calendar tasks. See examples below.
`calendar_set`	`--`	Yes	The `calendar_set tag` is used to set multiple calendar tasks. See examples below.
`calendar_delete`	`--`	Yes	The `calendar_delete` tag is used to delete all calendar tasks. See examples below.
	`task_id`	No	ID of the task. To delete all 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 to 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 the device supports.
	`free_count`	Yes	The unused number of calendar tasks the device supports.

Table 2.42: Calendar tags

The child tags for the `calendar_report` and `calendar_set` message are described in Table 2.43.

XML tag	Attribute	Required	Description
`task`	--	Yes*	The task will execute at 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 2.43: Calendar child tags

2.2.13.2 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
To get a specific calendar task, a `calendar_get` message has to be sent to the device with a task ID. See XML 2.62 for an example of this 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 2.62: Get the calendar task with task ID 1

Get all calendars
To get all calendar tasks, a `calendar_get` message has to be sent to the device without a task ID. See XML 2.63 for an example of this 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 2.63: 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 2.64 for an example of this 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 2.64: 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 2.65 for an example of this 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 2.65: Report all calendar tasks

Set a specific calendar task
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 2.66 for an example of this 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 2.66: Set the calendar task with ID 1

Set multiple calendar tasks

To set multiple calendar tasks, 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 2.67 for an example of this 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 2.67: Set multiple calendar tasks

Delete a specific calendar task

To delete a specific calendar task, a `calendar_delete` message has to be sent to the device with a task ID. See XML 2.68 for an example of this 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 2.68: Delete the calendar task with ID 1

Delete all calendar tasks

To delete all calendar tasks, a `calendar_delete` message has to be sent to the device without a task ID. See XML 2.69 for an example of this 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 2.69: Delete all calendar tasks

2.2.14 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 and by 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.

2.2.14.1 Tag description

The message tag for the state machine is described in Table 2.44 on the next page.

XML tag	Attribute	Required	Description
statemachine_get	--	Yes	To get states\transactions of the state machine. See examples below.
	statemachine_id	No	ID of the state machine. To get all state machines, omit statemachine_id.
statemachine_report	--	Yes	To report states\transactions of the state machine. See examples below.
statemachine_set	--	Yes	To set states\transactions of the state machine. See examples below.
statemachine_delete	--	Yes	To delete states\states 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	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	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	To set the current state of the state machine. See examples below.
statemachine_get_memory	--	Yes	To request the state machine memory information from a device.
statemachine_report_ memory	--	Yes	To report the state machine memory information from a device.
	count	Yes	Maximum number of state machines the device supports.
	free_count	Yes	Unused number of state machine the device supports.
statemachine_get_state_ memory	--	Yes	To request the state machine state memory information from a device.
statemachine_report_ state_memory .	--	Yes	To report the state machine state memory information from a device.
	count	Yes	Maximum number of state machine states the device supports.
	free_count	Yes	Unused number of state machine states the device supports.

Table 2.44: State machine tags

The child tags for the `statemachine_report` and `statemachine_set` message are described in Table 2.45.

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

Table 2.45: State machine message child tags

The child tags for the state machine message are described in Table 2.46.

XML tag	Attribute	Required	Description
`state`	`--`	Yes	The state that 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 2.46: State machine child tags

The child tags for the state are described in Table 2.47.

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 that the state machine will enter when this transaction is executed.

Table 2.47: State child tags

2.2.14.2 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
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 2.70 for an example of this 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 2.70: Get the state machine with ID 1 and its state with ID 2

Get a specific state machine

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 2.71 for an example of this 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 2.71: Get the complete state machine with ID 1

Get all state machines

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 2.72 for an example of this 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 2.72: 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 2.73 for an example of this 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_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 2.73: Report the state machine with ID 1 and its state with ID 2

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 2.74 for an example of this 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_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 2.74: Report the state machines with ID 1 and all its states

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 2.75 for an example of this 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_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 2.75: 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

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 2.76 for an example of this 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 2.76: Set the state machine with ID 1 state 1

Set a specific state machine

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 2.77 for an example of this 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 2.77: Set multiple states in the state machine with ID 1

Set multiple state machines

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 2.78 for an example of this 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 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 2.78: Set multiple states in the state machine with ID 1 and 2

Delete a specific state machines specific state

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 2.79 for an example of this 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_delete statemachine_id="1" state_id="2"/>
  </device>
</network>
```

XML 2.79: Delete the state with ID 2 from the state machine with ID 1

Delete a specific state machine

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 2.80 for an example of this 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_delete statemachine_id="1"/>
  </device>
</network>
```

XML 2.80: Delete the state machines with ID 1

Delete all state machines

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 2.81 for an example of this 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_delete/>
  </device>
</network>
```

XML 2.81: Delete all state machines

Get the current state

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 2.82 for an example of this 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_state statemachine_id="1"/>
  </device>
</network>
```

XML 2.82: Get the current state of the state machine

Report the current state

To report the current state of a state machine, the `statemachine_report_state` message has to be sent from the device with a state machine ID and its current state. See XML 2.83 for an example of this 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_report_state>
      <statemachine_state statemachine_id="1">3</statemachine_state>
    </statemachine_report_state>
  </device>
</network>
```

XML 2.83: Report the current state of the state machine

2.2.15 Firmware Update

The Firmware Update message is used to update the firmware on the devices.

2.2.15.1 Tag description

The message tag for the Firmware Update is described in Table 2.48.

XML tag	Attribute	Required	Description
`firmware_init`	`--`	Yes	Initialize the Firmware Update process.
	`size`	Yes	The size of the new firmware.
	`firmware_id`	Yes	The 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 checksum matches.
`firmware_report`	`--`	Yes	Reports the status of the last firmware update message.
	`status`	Yes	The status of the 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 2.48: Firmware Update tags

The child tags for the `firmware_data` message are described in Table 2.49.

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

Table 2.49: Firmware_data message child tags

2.2.15.1.1 Status

The valid options for the `status` attribute in the `firmware_report` are listed in Table 2.50.

Status ID	Status type
1	OK
2	Not Initialized
3	Size is too big
4	Checksum Error in received data
5	Data Overflow
6	Wrong Offset
7	Chunk size is too big
8	Data is missing
9	Chunk size is too small
10	Firmware Package blocked by Application

Table 2.50: Firmware Update status type

2.2.15.2 Examples

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

Initialize the firmware

To initialize the firmware, a `firmware_init` message has to be sent to the device with a size and a firmware ID. See XML 2.84 for an example of this 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 2.84: Initialize the firmware

Send a chunk of the firmware image

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 2.85 for an example of this 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 2.85: A chunk of the firmware image

Start the firmware update

To start the firmware update, a `firmware_update_start` message has to be sent to the device with a checksum. See XML 2.86 for an example of this 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 2.86: Start Firmware Update

Report the status of the last firmware update

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 2.87 for an example of this 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 2.87: Firmware report

Get the firmware information

To get the firmware information, a `firmware_information_get` message has to be sent to the device. See XML 2.88 for an example of this 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 2.88: Get the firmware information

Report the firmware information

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 2.89 for an example of this 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 2.89: Firmware information report

2.2.16 Status

The status is used to report information from a device.

2.2.16.1 Tag description

The message tag for the status is described in Table 2.51.

XML tag	Attribute	Required	Description
`status_report`	`--`	Yes	This tag is used to report a status.
	`type_id`	Yes	The type of the status. See Table 2.52.
	`code`	Yes	The code for the status.
	`level`	Yes	The level of the status. See Table 2.53.
	`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 the device. See Table 2.53.
`status_report_level`	`--`	Yes	This tag is used to report the status level from a device.
	`level`	Yes	The new status level for the device. See Table 2.53.

Table 2.51: Status tags

The valid types for the status are described in Table 2.52.

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 2.52: Status type

The valid level for the status level is described in Table 2.53.

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

Table 2.53: Status level

The device description code is described in Table 2.54.

Code	Name	Description
11	Set wrong ID	Trying to set a read-only value.
12	Wrong size of Channel Map	Trying to set a channel map that does not contain 4 channels.
13	Missing Synchronization channels	Trying to set a channel map without the required synchronization channels.

Table 2.54: Status device description code

The value description code is described in Table 2.55.

Code	Name	Description
1	Get Wrong ID	Trying to get a value description with an invalid ID.
2	Set Wrong ID	Trying to add a virtual value description with an invalid ID.
3	Delete Wrong ID	Trying to delete a virtual value with an 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 an invalid step.

Table 2.55: Status value description code

The value code is described in Table 2.56.

Code	Name	Description
1	Get Wrong ID	Trying to get a value with an invalid ID.
2	Set Wrong ID	Trying to set a value with an 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	Cannot read write only	Trying to read a write-only value.
16	Cannot write read only	Trying to write a read-only value.

Table 2.56: Status value code

The partner information code is described in Table 2.57.

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 message to a partner but there was no reply.
13	Set wrong type	Trying to set a partner with an unsupported info type.
14	Group in group not allowed	Trying to add a group to a group.
15	Too many partners in group	Trying to add more partners to a group then allowed.

Table 2.57: Status partner information code

The action code is described in Table 2.58.

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 2.58: Status action code

The calculation code is described in Table 2.59.

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 2.59: Status calculation code

The timer code is described in Table 2.60.

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 2.60: Status timer code

The calendar code is described in Table 2.61.

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 2.61: Status calendar code

The state machine code is described in Table 2.62.

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

Table 2.62: Status state machine code

The firmware update code is described in Table 2.63.

Code	Name	Description
11	Failed to upgrade	Failed to upgrade firmware from boot loader.

Table 2.63: Status firmware update code

The configuration code is described in Table 2.64.

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 2.64: Status configuration code

The system code is described in Table 2.65.

Code	Name	Description
11	No NTP	Failed to synchronize with the NTP Server.
12	Awake	Device was woken up and is now awake.
20	Hardware Fail - Dataflash	The device has detected a problem with the dataflash.

Table 2.65: Status system code

2.2.16.2 Examples

This section shows XML examples of how to use the `status_report`, `status_get_level`, `status_set_level` and `status_report_level` message.

Report a status
To report a status, a `status_report` message has to be sent from the device. See XML 2.90 for an example of this 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 2.90: Status report

Get a status level

To get the status level, a `status_get_level` message has to be sent from the device. See XML 2.91 for an example of this 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 2.91: Status get level

Report a status level

To report the status level, a `status_report_level` message has to be sent from the device. See XML 2.92 for an example of this 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 2.92: Status report level

Set a status level

To set the status level, a `status_set_level` message has to be sent to the device. See XML 2.93 for an example of this 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 2.93: Set status level

2.2.17 Configuration

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

2.2.17.1 Tag description

The message tag for the configuration is described in Table 2.66.

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 2.67.
`config_mode_set`	`--`	Yes	This tag is used to set the configuration mode on a device.
	`mode`	Yes	The configuration mode. See Table 2.68.

Table 2.66: Configuration tags

The valid values for the configuration status is described in Table 2.67.

Status	Name	Description
0	Idle	When no configuration is started.
1	Started	When a configuration is started.

Table 2.67: Configuration status

The valid configuration modes is described in Table 2.68.

Status	Name	Description
0	Rollback	Rollback any configuration changes.
1	Save and Reset	Save any configuration changes and reset the state machine states.
2	Save and Preserve	Save any configuration changes and do not change the state machine states.
3	Set Default	Clear the configuration and set the default configuration.
4	Clear	Clear the configuration.

Table 2.68: Configuration mode

2.2.17.2 Examples

This section shows XML examples of how to use the `configuration_status_get`, `configuration_status_report` and `configuration_mode_set` message.

Get configuration status
To get the configuration status, a `configuration_status_get` message has to be sent to the device. See XML 2.94 for an example of this 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 2.94: 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 2.95 for an example of this 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 2.95: Report configuration status

Set configuration mode
To set the configuration mode, a `configuration_mode_set` message has to be sent to the device. See XML 2.96 for an example of this 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 2.96: Set configuration mode

2.3 User stories

This section describes different user stories. Each subsection contains a user story, an illustration as well as examples of how to describe the scenario in XML.

2.3.1 Remote control with wall switch for controlling multiple devices

An individual comes home from work and would like to turn on the light and the TV.

- He/she pushes the wall switch that is connected to the electrical outlets of the light and the TV.
  As a result all appliances connected to this switch (i.e., the light and the TV) are turned on.

- Later that evening, when he/she is getting ready for bed, he/she pushes the wall switch again.
  As a result all appliances are turned off.

2.3.1.1 Illustration

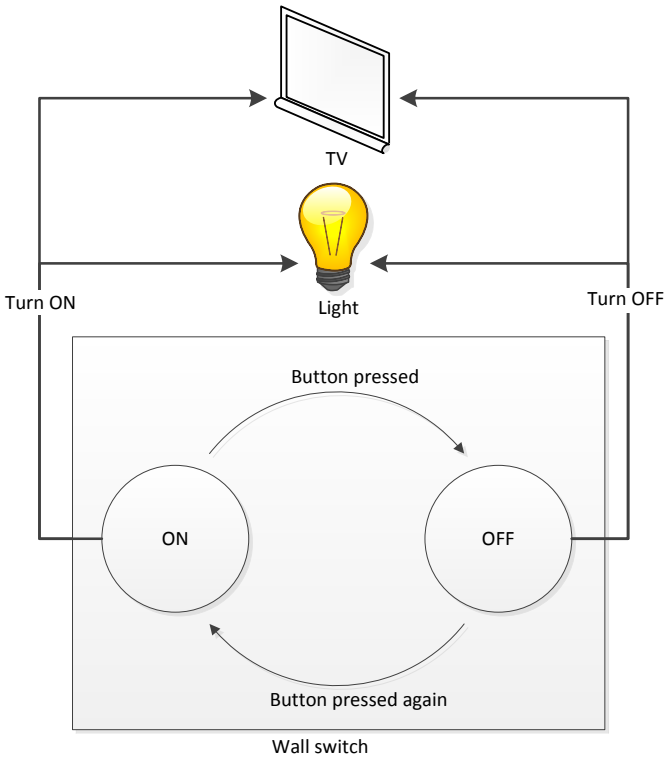

Figure 2.5: Remote control with light and TV

2.3.1.2 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 2.97: Calculation for User Story 1

```
<?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 2.98: Action for User Story 1

```
<?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 2.99: State Machine for User Story 1

2.3.2 Temperature control with calendar tasks

The temperature control in the house is set to 16 degrees Celsius at nighttime.

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

- In the late afternoon, when she/he comes home at 18:00, the temperature should be at an ambient temperature of 21 degrees Celsius. Finally, when she/he goes to bed at 22:00, the temperature should go to a steady temperature of 16 degrees Celsius during the night.

2.3.2.1 Illustration

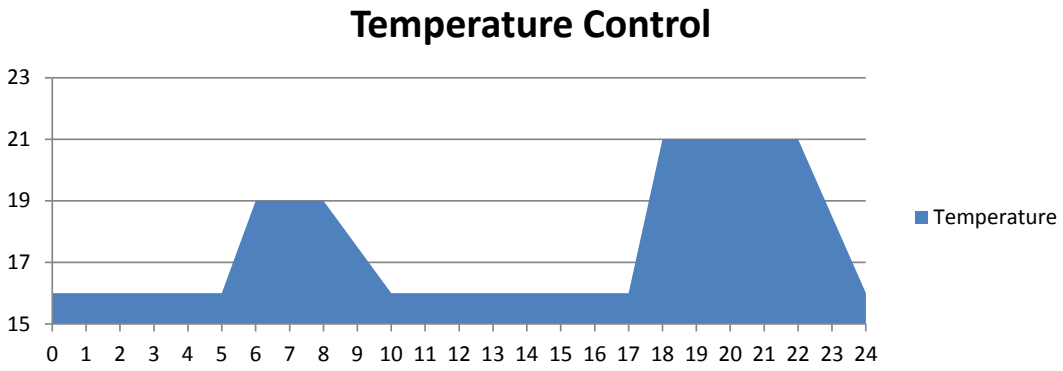

Figure 2.6: Temperature control with calendar tasks

2.3.2.2 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 2.100: Action for User Story 2

```
<?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 2.101: Calendar for User Story 2

2.3.3 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. This is detected by the system. The thermostat detects the temperature drop in the room, but allows the temperature it because of the open windows.

- After 5 minutes he/she closes the window again. As a result the temperature should rise and then stay at the abovementioned 21 degrees Celsius.

2.3.3.1 Illustration

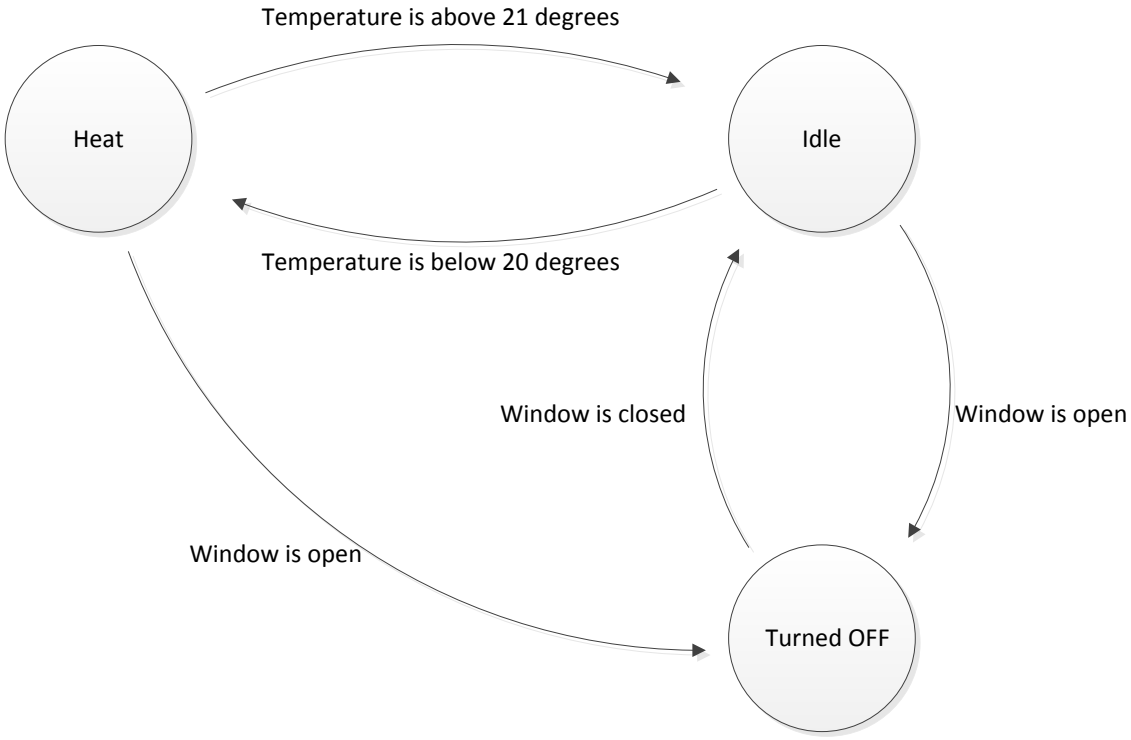

Figure 2.7: Temperature control with window and temperature sensor

2.3.3.2 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 2.102: Calculation for User Story 3

```
<?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 2.103: Action for User Story 3

```
<?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 2.104: State Machine for User Story 3

2.3.4 Light control with movement and luminance sensor

Depending on the daylight the movements of a person will turn the interior light on or off.

- An individual enters the living room while there is sufficient natural light. This movement is detected by the system. The System does not turn on the interior light as it is not necessary.

- In the evening when it is dark outside, the individual enters the living room again. This movement is detected by the system. The system turns on the interior light as it is necessary.

- As soon as the individual leaves the room and the system does not detect any movement in the room for a duration of 5 minutes, it will turn the lights off.

2.3.4.1 Illustration

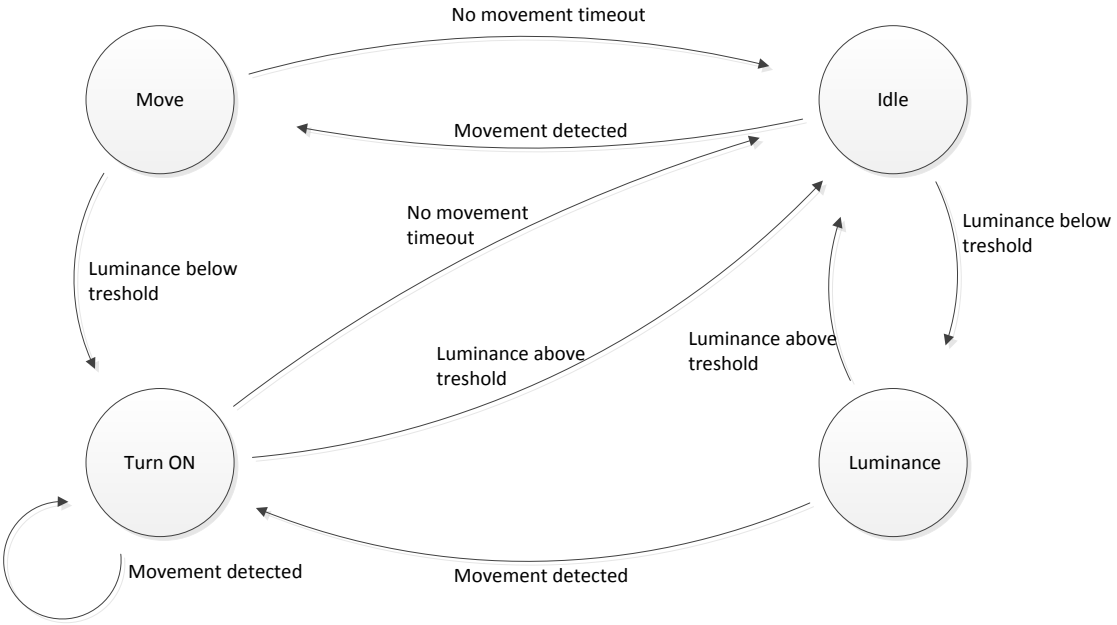

Figure 2.8: Light control with movement and luminance sensor

2.3.4.2 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 2.105: Calculation for User Story 4

```
<?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 2.106: Action for User Story 4

```
<?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 2.107: State Machine for User Story 4

```
<?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 2.108: Timer for User Story 4

2.3.5 Washing machine control

Before going to work, an individual prepares the washing machine by putting in the laundry, laundry detergent and, optionally, fabric softener.

- He/she wishes that the laundry is done when he/she comes back home.
- He/she sets the washing program to wool, and informs the system that he/she will be home at 18:00. Based on this time specification, the system determines the best start time for the washing cycle so that the laundry will be ready when the individual comes home.
- Later that day, he/she comes home earlier and starts the washing machine manually, thereby overruling the starting time set by the system.

2.3.5.1 Illustration

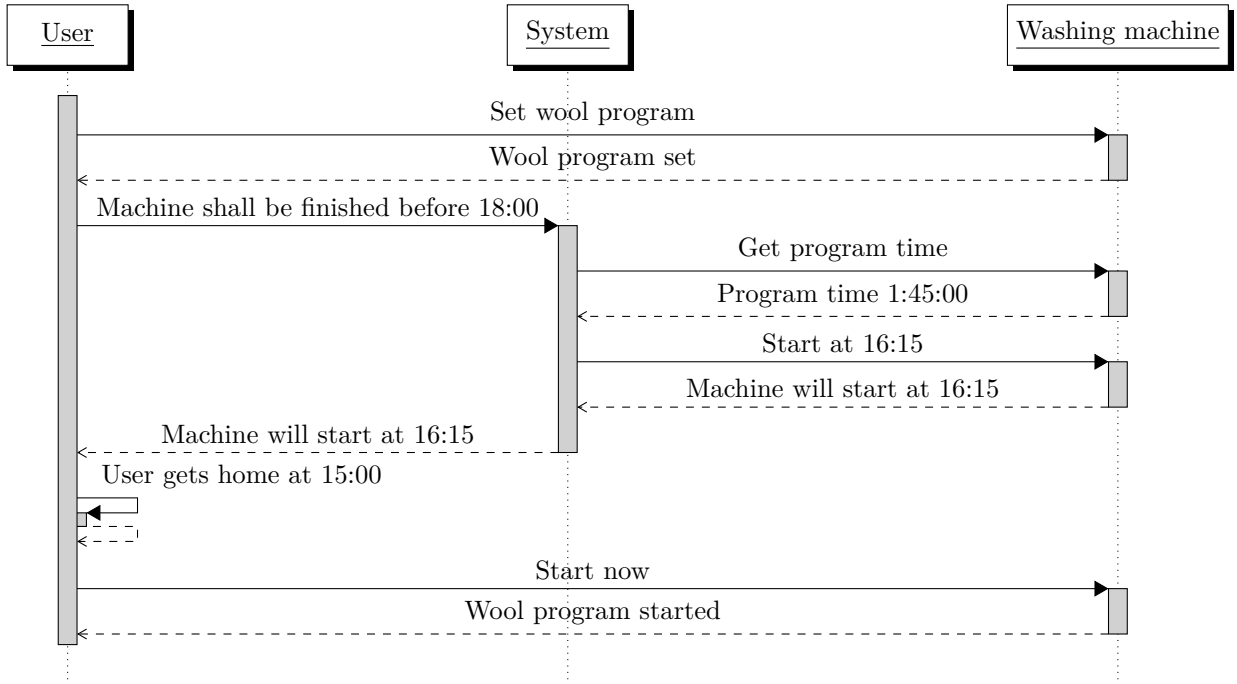

Figure 2.9: Washing machine control

2.3.5.2 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 2.109: 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 2.110: 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 2.111: Calendar for User Story 5

2.3.6 Electricity pricing

The electricity supplier of a household sends pricings to the electric meter on a regular daily basis.

This information is used to reduce the electricity bill. The reduction is done by performing actions that consume a lot of electricity and therefore cause greater expanses in periods with low pricing, e.g. at nighttime or midday on workdays.

2.3.6.1 Illustration

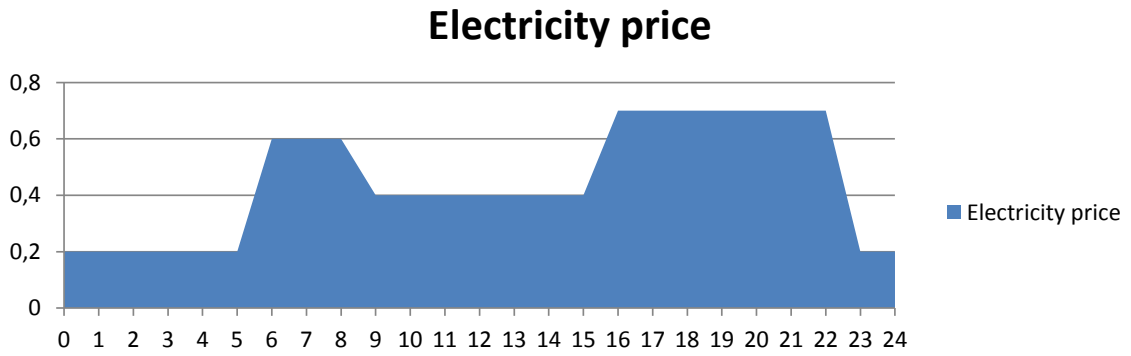

Figure 2.10: Electricity pricing

2.3.6.2 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 2.112: Action for User Story 6

```
<?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 2.113: Calendar for User Story 6

Chapter 3

Appendix 1: Lists

3.1 List of XSDs

3.1.1 Phy XSD

```
<?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 3.1: Phy XSD

3.1.2 Mac 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 3.2: 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 3.3: Mac XSD 2/2

3.1.3 Network Management XSD

```
<?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 3.4: Network Management XSD

3.1.4 Public Key XSD

```
<?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 3.5: Public Key XSD

3.1.5 Service Description XSD

```
<?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 3.6: Service Description XSD

3.1.6 Memory Information XSD

```
<?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 3.7: Memory Information XSD

3.1.7 Device Description XSD

```
<?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 3.8: Device Description XSD

3.1.8 Value Description XSD

```
<?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 3.9: 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 3.10: Value Description XSD 2/2

3.1.9 Value XSD

```
<?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 3.11: Value XSD

3.1.10 Partner Information XSD

```
<?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 3.12: 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 3.13: Partner Information XSD 2/2

3.1.11 Action XSD

```
<?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 3.14: 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 3.15: Action XSD 2/2

3.1.12 Calculation XSD

```
<?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 3.16: 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 3.17: Calculation XSD 2/2

3.1.13 Timer XSD

```
<?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 3.18: 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 3.19: Timer XSD 2/2

3.1.14 Calendar XSD

```
<?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 3.20: 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 3.21: Calendar XSD 2/2

3.1.15 State Machine XSD

```
<?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 3.22: 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 3.23: State Machine XSD 2/2

3.1.16 Firmware Update XSD

```
<?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 3.24: 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 3.25: Firmware Update XSD 2/2

3.1.17 Configuration XSD

```
<?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 3.26: Configuration XSD

3.1.18 Status XSD

```
<?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 3.27: Status XSD

List of Tables

List of XMLs

List of Figures

Chapter 4

Appendix 2: Changes between document versions

4.1 From Version 1.8 Draft to Version 1.12

Here you will find changes and other differences like added or deleted text that were made between the different released versions of this Lemonbeat document. Version 1.8 Draft was released in August 2016, and version 1.12 in January 2017. Versions in between were internal versions only.

4.1.1 General changes

Layout changes

- Mapping of figures/tables and text optimized.
- Text structure changes (e.g., changing longer paragraphs into lists etc.).
- Consistent formatting of code segments in typewriter font.

Linguistic changes

- Linguistic corrections (grammar, typos, punctuation, syntax etc.).
- Deleted internal comments like FIXme etc.

Terminology changes

- ‘‘Application layer‘‘ replaced by ‘‘Lemonbeat smart Device Language‘‘ and a corresponding explanation where appropriate.

4.1.2 List with individual changes

Chapter	Title	Table/figure	Difference
	Title page		New document title (“Lemonbeat smart Device Language“) to better reflect the content.
2.1	**Application layer**		
2.2.1	NTP		Text rephrased for a better understanding.
2.2	**Protocol description**		New chapter title (“Lemonbeat smart Device Language description“) to better reflect the content.
2.2.1	Services		Note added that currently only UDP is supported.
2.2.1 2.2.2 2.2.3 2.2.3.2 2.2.3.3 2.2.3.4 2.2.3.5 2.2.3.10	Services Network management Device Configuration Partner Service Timer Service Calendar Service Action Service Firmware Update Service		Sections rephrased for a better understanding.
2.3	**XML description**		
2.3.1	Network and device	old: Table 2.6 new: Table 2.4	Unit milliseconds addded to `go_to_sleep` entry.
2.3.2 2.3.2.2	Network Management Inclusion data		Added “AES“ to “controller key“.
2.3.4 2.3.4.1	Service Description Management Tag description	 old: Table 2.12 new: Table 2.10	 Deleted asterisk and explanation to asterisk. Reason: the content is already mentioned in front of the table.
2.3.5	Memory information		Text rephrased for a better understanding.

Chapter	Title	Table/figure	Difference
2.3.6	Device Description		
2.3.6.1	Tag description	old: Table 2.19 new: Table 2.17	Entry ‘‘Wakeup Offset‘‘: Changed ‘‘RX‘‘ to ‘‘RX-active‘‘ and added information ‘‘Not needed in the current version‘‘ in description.
		old: Table 2.21 new: Table 2.19	Entry ‘‘Radio Mode‘‘: Added information that only UDP is supported for Radio Mode ID 1 (Wake-on-Radio) and 2 (Wake-on-Event).
	Section ‘‘SGTIN‘‘		Replaced wrong long name with correct long name (also in rest of the document). Rephrased for a better understanding. Deleted information about the different items that are part of a SGTIN. Reason: these can be found in the referenced standard.
	Section ‘‘Channel map‘‘		Added missing table number in reference.
2.3.8 2.3.8.2	Value Example		Rephrased first paragraph under ‘‘Example‘‘ for a better understanding.
	Section ‘‘Get Log values for a specific value‘‘		Deleted. Reason: Not implemented.
	Section ‘‘Report log values for a specific value‘‘		Deleted. Reason: Not implemented.

Chapter	Title	Table/figure	Difference
2.3.9 2.3.9.1	Partner Information Tag description	 old: Table 2.31 new: Table 2.27	 Minor addition to `free_count` attribute entry.
		old: Table 2.34 new: Table 2.30	Additional information in entries "Wakeup Interval" and "Wakeup Offset".
2.3.9.2	Using partner information to control devices supporting sleep mode		New subsections with the following topics: Handling Wake-on-Radio (WoR) devices Handling Event-Listening devices
2.3.10 2.3.10.1	Action Tag description	 old: Table 2.39 new: Table 2.35	 Added some more information about UDP and TCP transport mode.
2.3.11 2.3.11.1	Calculation Tag description	 old: Table 2.42 new: Table 2.38	Rephrased for a better understanding. Added information that attributes `constant_string` and `constant_hexBinary` are not supported.
2.3.14 2.3.14.2	State Machine Examples		Section "Get the current state": rephrased for a better understanding.
2.3.16 2.3.16.1	Status Tag description	 old: Table 2.58 new: Table 2.54	 Changed table content to a more precise and correct one.
		old: Table 2.61 new: Table 2.57	Added two more table entries.
2.3.17 2.3.17.1	Configuration Tag description	 old: Table 2.71 new: Table 2.67	 Table entry to status O: rephrased for a better understanding.
2.4	User stories		Generally rephrased for a better understanding.

4.2 From Version 1.12 to Version 1.13

Here you will find changes and other differences like added or deleted text that were made between the different released versions of this Lemonbeat document. Version 1.12 was released in January 2017, and version 1.13 is the current version. In this version you will find mainly layout changes.

4.2.1 General changes

Layout changes

- Further mapping of figures/tables and text optimized.
- Smaller margin for readability reasons. This lead to general layout and pagination changes.
- Some changes of the order of chapters and sections. Reason: for a more logical order. For example, the chapter ‘‘XML description‘‘ is now in front of the chapter ‘‘User Stories‘‘.

Terminology changes

- Changed ‘‘Lemonbeat Protocol‘‘ to ‘‘Lemonbeat Protocol Stack‘‘ to be more precise.

4.2.2 List with individual changes

Chapter	Title	Table/figure	Difference
	Title page		New document title (“Lemonbeat smart Device Language“) to better reflect the content.
old: **2.1** new: n/a	**Application layer** n/a		Deleted. Reason: not needed.
old: 2.1.1 new: 2.1.3	NTP		Restructured.
old: 2.2.1 new: 2.1.1	old: Services new: Service types and related ports		New introduction added.
old: 2.2.2 new: 2.1.2	Network management		Mapping of figures/tables and text optimized.
old: n/a new: 2.2.6.2	Device description types	 Table 2.17	Added subsection title and introduction. New entries for IDs 22 to 27.
old: **2.3** new: **2.2**	**XML description**		
old: 2.3.1 new: 2.2.1	Network and device		Changed “amount of time“ to “time span“ to be more precise.
old: 2.3.9 new: 2.2.9	Partner Information		Rephrased first paragraph a bit for a better understanding.
old: 2.3.14 new: 2.2.14 old: 2.3.14.1 new: 2.2.14.1	State Machine Tag description	 Table: 2.44	 Tag description for State Machine shortened and rephrased for consistency reasons.
old: **2.4** new: **2.3**	**User Stories**		Whole chapter: Some changes to text structure (bulleted lists etc.) and some rephrasing for a better understanding.

lemonbeat GmbH
Deutsche Str. 5
D-44339 Dortmund
Phone: +49 (0)231 – 586 937 0
E-mail: info@lemonbeat.com