sub
Hochschule Offenburg
offenburg.university

ivESK
Institut für verlässliche
Embedded Systems und
Kommunikationselektronik

Offenburg University of Applied Sciences
Institute of reliable Embedded Systems and Communication Electronics (ivESK)

emb::6 Serial API Specification

Project: emb::6 Serial API Specification
Version: 0.18
Date: 31.01.2018

Authors

Manuel Schappacher [manuel.schappacher@hs-offenburg.de]

History

Version	*Author(s)*	*Date*	*Summary of changes*
0.1	M. Schappacher	20.07.2016	Initial Creation
0.2	M. Schappacher	24.10.2016	Initial Description of the general API
0.3	M. Schappacher	07.12.2016	Initial description of the LWM2M API
0.4		28.02.2017	- Removed INIT state and the commands related to that. - Changed command IDs for setting status. - Changed Status IDs - Changed startup sequence - Changed configuration sequence
0.5	M. Schappacher	07.03.2017	- Fixed length field values in examples - Fixed size of return codes (from 16 to 8-Bit) - Added chapter “Next Layers”
0.6	M. Schappacher	08.03.2017	- Fixed typos
0.7	M. Schappacher	09.03.2017	- Added descriptions for LWM_RES_RD_RSP, LWM_RES_WR_REQ, LWM_RES_WR_RSP
0.8	M .Schappacher	09.03.2017	- Changed document structure - Updated sequence diagrams
0.9	M. Schappacher	09.03.2017	- Added UDP API section - Minor corrections - Added templates for Read/Write/Observe for LWM2M Instances
0.10	M. Schappacher	10.03.2017	-
0.11	M. Schappacher	01.04.2017	- Minor Corrections - Fixed wrong size for OBJ ID in LWM2M example
0.12	M. Schappacher	28.06.2017	- Added description of Resource-Commands
0.13	M. Schappacher	13.07.2017	- Added size field for adding LWM2M resources
0.14	M. Schappacher	22.09.2017	- Added function to reset LWM2M layer - Initial value can be set when creating a resource - Resource Access Modifier is mandatory
0.15	M. Schappacher	17.11.2017	- Added Instance Read and Write functions - Added LWM2M Timeout Configuration
0.16	M. Schappacher	22.11.2017	- Added Length parameter to Instance Write and Response functions
0.17	M. Schappacher	27.11.2017	Small corrections
0.18	M. Schappacher	31.0.1.2017	Additional inverted length field

Abbreviations

Abbreviation	*Description*
C	Command
CAS	Configuration API Set
CMD	Command Field
CRC	Cyclic Redundancy Check
CSMA-CA	Carrier Sense Multiple Access – Collision Avoidance
D	Device
GAP	General API Set
H	Host
HSO	Offenburg University of Applied
ivESK	Institute of Reliable Embedded Systems and Communication Electronics
LWM2M	Lightweight M2M
NBO	Network Byte Order
R	Response
SFD	Start of Frame Delimiter

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduction

emb::6 is a scalable C-based 6LoWPAN stack for embedded devices developed by the Institute of Reliable Embedded Systems and Communication Electronics (ivESK) at Offenburg University of Applied Sciences (HSO). Originally derived from Contiki, ivESK made several adaptations such as the removal of proto-threads. It follows a strict layer based architecture with a modular software design allowing its usage even on very restricted devices. One of the typical use cases of emb::6 is the Home and Building Automation sector as shown in Fig. 1.

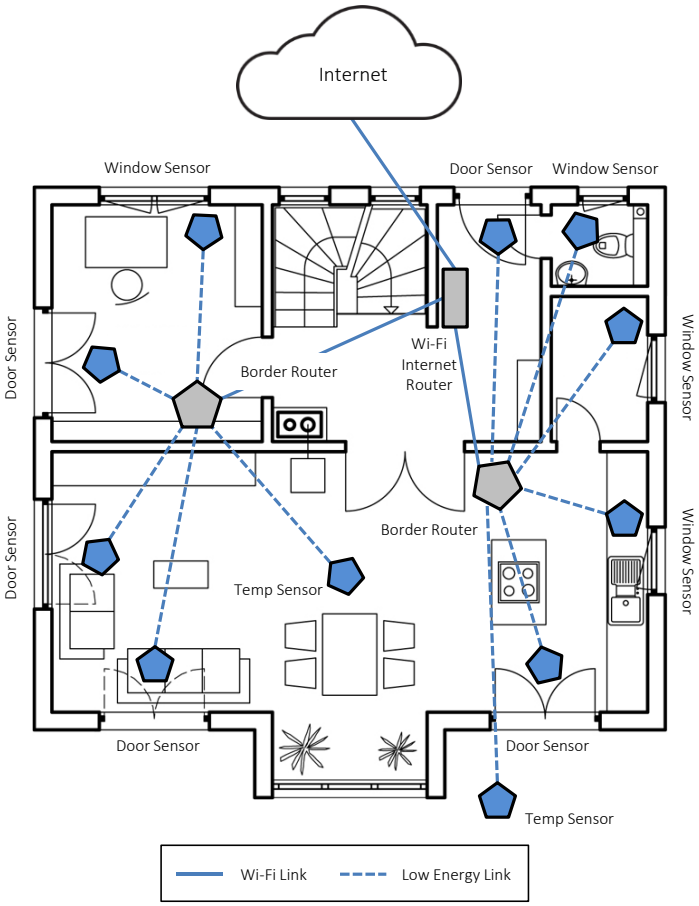

Fig. 1: Typical use case of emb::6 within Home and Building Automation Applications

1.1 Scope

This document described the Serial API Application and its interfaces. It allows using and controlling an embedded device running emb::6 from a separate controller. Therefore, the embedded device that is running emb::6 acts as a communication module or modem.

1.2 LWM2M basics

TBD

2 General Description

This chapter describes some basic requirements and specifications required for the Serial API and its usage. This includes its role within the emb::6 stack as well as some basic definitions.

2.1 Serial API

The Serial API acts as an interface for external controllers that allow them to control the communication controller running emb::6. Therefore, a separate Application within the Demo Layer, called SAPI represents the Serial API implementation as shown in Fig. 2.

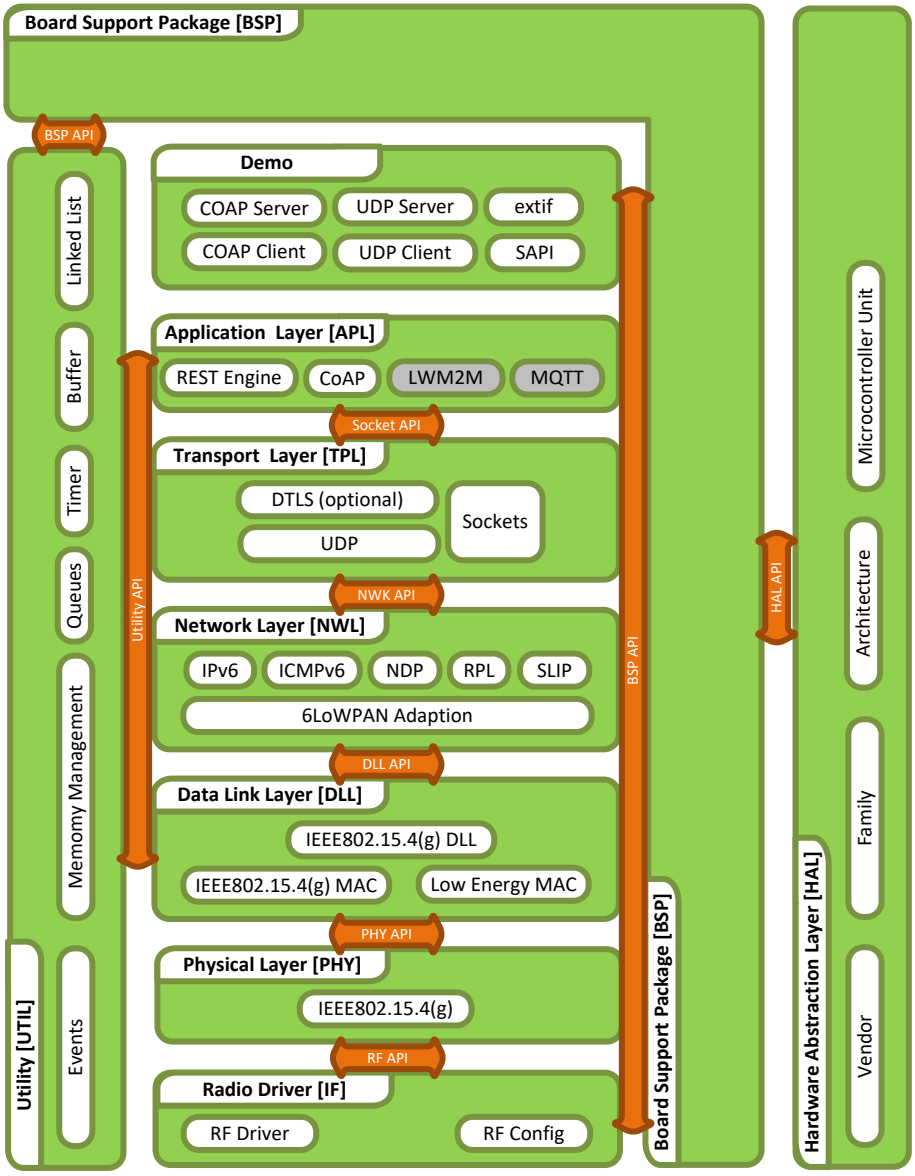

Fig. 2: Stack architecture including Serial API application

2.2 Byte Order

Since the Serial API make use of 16-Bit and 32-Bit values, the Serial API must specify an according byte order to guarantee platform compatibility. Therefore, the Serial API uses Network Byte Order (NBO).

2.3 General Command Flow

In general, the host controller always starts a serial communication (e.g. sending a request). However, the module also can call events (e.g., if the communication module received a frame). Furthermore, the

communication module should reply all commands transmitted by the host controller; otherwise, this could indicate a problem.

2.3.1 Timeouts

As described, the communication flow follows a request/response paradigm for most of the procedures. To recognize errors within the communication flow, the specification applies timeouts to expected responses. If not specified further, each command shall have a common timeout of tot_{CMD} = 50ms.

2.4 Base Frame Format

The following section describes the base frame format of the communication frames. The Serial API encapsulates all data within a common frame format as shown in Fig. 3. A frame splits up into a header, a data part and a footer. The header starts with a Start of Frame Delimiter (SFD) followed by a Length field. The type field describes the data following to the header. Finally, a checksum is included at the end of the frame.

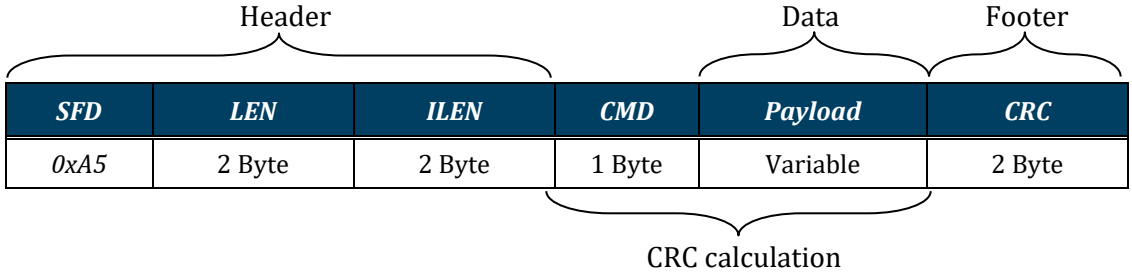

Fig. 3: Base frame format of the Serial API

2.4.1 Start of Frame Delimiter

The Start of Frame Delimiter (SFD) introduces a new frame. It has a fixed value of *0xA5*. Every frame shall start with the SFD. The presence of the Length field included within the base frame format also allows using the SFD value within the payload of a frame.

2.4.2 Length

The Length field (LEN) describes the length of a frames payload. It does not consider the SFD, LEN, ILEN and CRC fields for the length calculation but includes the CMD field. An additional inverted Length field was added for error detection. It holds the inverted value of the Length field.

2.4.3 Command

The Command Field (CMD) describes the type of command transmitted using the Serial API. Ch. 3.4 list and describes the type of commands that are available. Some of the commands are available in both directions, whereas some are unidirectional only.

2.4.4 Payload

The payload field holds the actual data depending on the value of the CMD fields. The payload field has a variable length. The actual length of the payload field can be determined by the length field.

2.4.5 Cyclic Redundancy Check

The Cyclic Redundancy Check (CRC) guarantees proper transmission and reception of a frame. Therefore, the base frame format uses a 16-Bit CRC with the polynomial *x16 + x13 + x12 + x11 + x10 + x8 +x6 + x5 +x2 + 1*. The initial value is zero; the final CRC is complemented.

List. 1 gives a sample implementation of the CRC procedure.

```
/*! x^16 + x^13 + x^12 + x^11 + x^10 + x^8 + x^6 + x^5 + x^2 + 1
 required when manually computing */
#define POLYNOMAL                    0x3D65U

uint16_t crc_calc(uint16_t i_init, uint8_t *pc_data, uint16_t i_len)
{
  /* Temporary variables */
  uint16_t i, j, c;
  /* Stores the current calculated crc value */
  uint16 t i crc;
  i crc = i init;

  for(i = 0U;i < i_len;i++)
  {
    c = pc data[i];
    c <<= 8U;
    i_crc ^= c;

    for(j = 0U;j < 8U;j++)
    {
      if(i crc & 0x8000U)
        i crc = (i crc << 1U) ^ POLYNOMAL;
      else
         i_crc <<= 1U;
    }
    i crc &= 0xFFFFU;
  }
  return i crc;
}
```

List. 1: CRC implementation

2.5 Access Methods

The LWM2M API as described in ch. 4 specifies different access methods for values namely polled access and stored access. When using polled access the device initiates a read request to the host to get the actual value every time a value was requested from a LWM2M server. When using stored values the device holds a local cop of the value that is reported to the LWM2M server on every request. To update this value the host has to update the value explicitly.

2.6 Type and Direction

Some commands (C), responses (R) or unsolicited messages (U) can be transmitted in both ways, where as some are used unidirectional only. Therefore, a device (D) refers to the communication module and the host (H) to the host controller. Tab. 1 shows the available directions.

Name	*Scope*		*Description*
	Host	*Device*	
H2D	Tx	Rx	Direction from host to device.
D2H	Rx	Tx	Direction from device to host.
BD	Tx/Rx	Tx/Rx	Bidirectional communication.

Tab. 1: Directions of communications

The highest bit of the command identifier represents the type of the command/response, whereas the high-bit set indicates a command and the high-bit cleared indicates a response.

3 General API Description

The General API Set (GAS) is responsible for handling and controlling the status and operation of the device. It provides functionality e.g. for starting and stopping a device as well as for configuring a device.

3.1 Status and Return Codes

Several commands and responses make use of status and return codes. Such codes rely on a 8-Bit variable. Tab. 2 lists all of the possible codes with their according values.

Name	*Value*	*Description*
OK	0x00	Describes a positive return value e.g. after a command was issued.
ERROR	0x01	A general error occurred during the operation (e.g. CRC error).
ERROR_CMD	0x02	The command is not valid or supported.
ERROR_PARAM	0x03	The parameters are invalid or not supported.
ERROR_SYNC	0x04	Invalid Sync Byte received.
ERROR_LEN	0x05	Error in the length verification detected.
ERROR_CRC	0x06	CRC missmatch.
STATUS_STOPPED	0x30	The device stopped its operation.
STATUS_STARTED	0x31	The device started its operation. In this state, the device is usually trying to access and connect to the wireless network.
STATUS_NETWORK	0x32	The device has access to the network and is capable to communicate. All the commands directed to higher layer shall fail if the device is not within this state.
STATUS_ERROR	0x3E	The device is in an error state.
STATUS_UDEF	0x3F	The device is in an undefined state. Usually a device enters this state after power-up. The host shall initialize a device that resides in this state.

Tab. 2: Status and return codes of the Serial API

The emb::6 stack starts automatically after power-up. This means it automatically goes to the *STATUS_STARTED* state and tries to connect to a network. To change the configuration it has to be stopped explicitly.

3.2 Error Codes

The *ERROR_GET* command allows retaining a more specific error code in case a device retains in its error state *STATE_ERROR*. Tab. 3 lists all the available error codes.

Name	*Value*	*Description*
ERROR_NO	0x00	No Error occurred so far.
ERROR_UNKNOWN	0x01	An unknown error occurred.
ERROR_FATAL	0xFF	Fatal error occurred.

Tab. 3: Error codes of the Serial API

3.3 Next Layers

The general API provides a so called *NEXT_LAYER* command. This command can be used to generate commands targeted to a specific layer of the serial API. All commands using such a command ID will

be passed directly to the according next layer handler and will not be processed by the general API. This allows to define independent API sets for higher layers.

3.4 Function Overview

The General API Set (GAS) is responsible for handling and controlling the status and operation of the device. It provides functionality e.g. for starting and stopping a device as well as for configuring a device. Tab. 4 shows the functions of the General API Set.

Name	*Type*	*Value*	*Direction*	*Description*
RET	R	0x00	BD	General response to acknowledge a command. A device/host shall issue this response for every command, in case no specific response exists. This is required to see if the command was received and accepted.
PING	C	0x10	BD	A host can use this device to check the availability of a device and vice versa.
CFG_SET	C	0x20	H2D	Set a configuration parameter. Tab. 7 lists the available configurations.
CFG_GET	C	0x21	H2D	Get a configuration parameter. Tab. 7 lists the available configurations.
CFG_RSP	R	0x22	D2H	Return a configuration parameter. Tab. 7 lists the available configurations.
DEVICE_STOP	C	0x30	H2D	Stop the communication module. Stops the operation of the communication module.
DEVICE_START	C	0x31	H2D	Start the communication module. The communication module tries to access and connect to the network.
STATUS_GET	C	0x40	H2D	Get the Status of the communication module. This is required for the sensor module to know when the communication is ready or if an error occurred.
STATUS_RET	R/U	0x41	D2H	Returns the status of the communication module. The device creates this response either when requested using the STATUS_GET command or automatically if the status of the communication module has changed. Status can be any of the status codes from Tab. 2
ERROR_GET	C	0x50	H2D	Obtain the last error of the communication module.
ERROR_RET	R	0x51	D2H	Returns the latest error of the communication module.
NEXT_LAYER	C/R/U	0xE0-0xFF	BD	Reserved for next layer operations. All commands with such an ID will be forwarded to the next layer (e.g. LWM2M). - UDP (0xE0) - LWM2M (0xE1)

Tab. 4: Functions of the General API Set

3.5 Function Descriptions

3.5.1 RET

The *RET* response is a bidirectional response used to signal the proper or invalid reception of a command frame. The *RET* response shall be sent for every command that does not specify an according answer. It follows the frame format as shown in Fig. 4.

LEN	*CMD*	*Payload*
		RET
2	1 Byte (Tab. 4)	1 Byte

Fig. 4: Frame format of the RET response

3.5.1.1 Parameters

Name	*Type*	*Description*
RET	ENUMERATION	Status can be any of the status codes from Tab. 2 without a prefix.

Tab. 5: Parameters of the RET response

3.5.2 PING

A host/device uses the *PING* command to check the availability of the device/host. After reception of this command the device or host replies with a *RET* response to indicate its availability.

It has no parameters and therefore zero length. The *PING* command does not assume a specific response. It uses the *RET* response from the device/host as the according Reply.

3.5.3 CFG_ (SET/GET_RET)

The configuration functions *CFG_ (SET/GET/RET)* provide the opportunity to change a device configuration. Each configuration parameter has its default value. The device will use that default value in case the host does not perform a specific configuration.

It follows the frame format as shown in Fig. 5. Depending on the type of the command, the device replies with an according answer. For the *CFG_SET* command the device replies with a regular *RET*, for the *CFG_GET* command the device replies with a *CFG_RET.*

LEN	*CMD*	*Payload*	
		CFG_ID	*CFG_VAL*
2+x	1 Byte (Tab. 4)	1 Byte	variable

Fig. 5: Frame format of the CFG frames

3.5.3.1 Parameters

Name	*Type*	*Description*
CFG_ID	uint8	ID used to identify the type of configuration to access. The ID can be any of those described in Tab. 7.
CFG_VAL	variable	The actual configuration to write, to obtain or indicated.

Tab. 6: Parameters of the CFG frames

3.5.3.2 Configurations

Name	*CFG-ID*	*Type*	*Default Value*	*Description*
MACADDR	0x00	char[8]	DE:AD:BE:EF:00:00	The device MAC address that will be used for the communication.*
PANID	0x01	uint16	0xCAFE	The device PANID that will be used for the communication.*
OPMODE	0x02	ENUMERATION	MODE1	The communication stack can run at different modes (c.f. [1]). This parameter allows changing the current operation mode.*
CHANNEL	0x03	uint8	0	The communication stack can run at different channels. This parameter allows changing the active communication channel.*

Tab. 7: General API configuration parameters

**Some configurations can only by applied if the device is in STOP state. In any other state, the device will either reject the request or apply it after the next initialization. Please check the return value of the configuration command to know about the status.*

3.5.4 DEVICE_STOP

The host issues the *DEVICE_STOP* command to the device to stop the communication module and the device disconnects from the network. Previous configurations remain until the next initialization request.

It has no parameters and therefore zero length. The *DEVICE_STOP* command does not assume a specific response.

3.5.5 DEVICE_START

The host issues the *DEVICE_START* command to the device to start the communication module and the device starts trying to access the network. The host shall perform configurations before starting a device. Otherwise, the device uses its default configuration.

It has no parameters and therefore zero length. The *DEVICE_START* command does not assume a specific response.

3.5.6 STATUS_GET

The *STATUS_GET* command retrieves the status of the device. On reception, the device will respond with its status.

The command has no parameters and therefore zero length. The *STATUS_GET* command expects an *STATUS_RET* response.

3.5.7 STATUS_RET

A device issues a *STATUS_RET* response either on reception of a *STATUS_GET* request or autonomously when its internal status has changed (e.g. if it successfully attached to a network and changed its status to *STATUS_NETWORK*). It follows the frame format as shown in Fig. 6.

LEN	*CMD*	*Payload*
		STATUS
2	1 Byte (Tab. 4)	1 Byte

Fig. 6: Frame format of the STATUS response

3.5.7.1 Parameters

Name	*Type*	*Description*
STATUS	ENUMERATION	Status can be any of the status codes from Tab. 2 using a STATUS_* Prefix

Tab. 8: Parameters of the STATUS response

3.5.8 ERROR_GET

A host can use the *ERROR_GET* command to retrieve the specific error code of a device that remains in *STATE_ERROR*.

It has no parameters and therefore zero length. The *ERROR_GET* command expects an *ERROR_RET* response.

3.5.9 ERROR_RET

A device issues an *ERROR_RET* response on reception of an *ERROR_GET* request. It indicates its current specific error when it is in STATE_ERROR. Otherwise, the response returns *ERROR_NO*. It follows the frame format as shown in Fig. 7.

LEN	*CMD*	*Payload*
		ERROR
2	1 Byte (Tab. 4)	1 Byte

Fig. 7: Frame format of the ERROR response

3.5.9.1 Parameters

Name	*Type*	*Description*
ERROR	ENUMERATION	Error can be any of the error codes from Tab. 2.

Tab. 9: Parameters of the ERROR response

4 UDP API Description

The serial API provides the functionality to transmit and receive raw UDP packets. Such packets do not have a specific format. They can be used as pure data transport mechanism.

4.1 Status and Return Codes

Several commands and responses make use of status and return codes. Such codes rely on a 8-Bit variable. Tab. 14 lists all of the possible codes with their according values.

Name	*Value*	*Description*
OK*	Tab. 2	Describes a positive return value e.g. after a command was issued.
ERROR*	Tab. 2	A general error occurred during the operation (e.g. CRC error).
ERROR_CMD*	Tab. 2	The command is not valid or supported.
ERROR_PARAM*	Tab. 2	The parameters are invalid or not supported.
ERROR_SOC	0x10	An invalid socket was given as parameter.
ERROR_LEN	0x11	An invalid length was given e.g. for data transmission.

Tab. 10: Status and return codes of the UDP API Set

**For some of the return codes the same values as for the general API is used. This holds true e.g. for the basic return codes such as for OK, invalid commands or invalid parameters.*

4.2 Error Codes

TBD

4.3 Function Overview

The serial API provides the functionality to transmit and receive raw UDP packets. Such packets do not have a specific format. They can be used as pure data transport mechanism. Therefore the API provides simple functions e.g. to create sockets and to transmit and received data through them. Tab. 11 shows the functions of the UDP API Set.

Name	*Type*	*Value*	*Direction*	*Description*
UDP_RET	C	TBD	R	Response to command. A device/host shall issue this response for every command, in case no specific response exists. This is required to see if the command was received and accepted.
UDP_INIT	C	TBD	H2D	Initialize the UDP communication. A host shall initiate this command before any other.
UDP_SOCK_CREATE	C	TBD	H2D	Create a new UDP socket with a specific destination address and port.
UDP_SOCK_CREATE_RET	R	TBD	H2D	Response to a previously initiated socket create command.
UDP_SOCK_DELETE	C	TBD	H2D	Delete a socket that was created previously.
UDP_SOC_SEND	C	TBD	H2D	Send Data via an existing socket.
UDP_SOC_RECV	U	TBD	H2D	Initiated from the device if data was received from a socket.

Tab. 11: Functions of the UDP API Set

4.4 Function Description

4.4.1 UDP_RET

The *UDP_RET* response is a bidirectional response used to signal the proper or invalid reception of a UDP command. The *UDP_RET* response shall be sent for every command that does not specify an according answer. It follows the frame format as shown in Fig. 8.

LEN	*CMD*	*CMD UDP*	*Payload*
			RET
3+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 11)	1 Byte

Fig. 8: Frame format of the UDP_RET response

4.4.1.1 Parameters

Name	*Type*	*Description*
RET	ENUMERATION	Status can be any of the status codes from Tab. 10.

Tab. 12: Parameters of the UDP_RET response

4.4.2 UDP_INIT

A host uses the *UDP_INIT* command to initialize the UDP module of the device. After reception of this command the device replies with a UDP_*RET* response to indicate the status of the initialization request.

It has no parameters and therefore zero length. The *UDP_INIT* command does not assume a specific response. It uses the *UDP_RET* response from the device as the according reply.

4.4.3 UDP_SOCK_CREATE

A host uses the *UDP_SOCK_CREATE* function to create a new socket. Such a socket is defined by its source and destination address and port which must be provided as parameter.

It follows the frame format as shown in Fig. 9. The device replies with an according *UDP_SOCK_CREATE_RET* response to indicate the ID of the socket that was created on success or an error in case no socket could be created.

LEN	*CMD*	*CMD UDP*	*Payload*		
			SRC_IP	*DST_IP*	*PORT*
20	1 Byte (NEXT_LAYER)	Tab. 11	16 Byte	16 Byte	2Byte

Fig. 9: Frame format of the UDP_SOCK_CREATE command

4.4.3.1 Parameters

Name	*Type*	*Description*
SRC_IP	char[16]	IP address of the source IP. The source IP is used to filter incoming messages. In case a specific address is given only messages from this address will berecived
DST_IP	char[16]	IP address of the destination.
PORT	uint16	Port of the destination.

Tab. 13: Parameters of the UDP_RET response

4.4.4 UDP_SOCK_CREATE_RET

TBD

4.4.5 UDP_SOCK_DELETE

TBD

4.4.6 UDP_SOC_SEND

TBD

4.4.7 UDP_SOC_RECV

TBD

5 LWM2M API Description

The LWM2M API Set (LAS) provides the main functionality regarding to device management and data exchange using the LWM2M layer of the emb::6 communication stack. To use the API a basic knowledge of the LWM2M paradigm is required. This is available from [XXX]. Tab. 16 shows the functions of the LWM2M API Set.

5.1 Status and Return Codes

Several commands and responses make use of status and return codes. Such codes rely on a 8-Bit variable. Tab. 14 lists all of the possible codes with their according values.

Name	*Value*	*Description*
OK*	Tab. 2	Describes a positive return value e.g. after a command was issued.
ERROR*	Tab. 2	A general error occurred during the operation (e.g. CRC error).
ERROR_CMD*	Tab. 2	The command is not valid or supported.
ERROR_PARAM*	Tab. 2	The parameters are invalid or not supported.
STATUS_STOPPED	0x30	The LWM2M module stopped its operation.
STATUS_STARTED	0x31	The LWM2M module started its operation. In this state, the device is usually trying to connect to a server.
STATUS_REGISTERED	0x32	The device is registered at a server.
STATUS_BOOT	0x33	Bootstrapping is executed.

Tab. 14: Status and return codes of the LWM2M API Set

**For some of the return codes the same values as for the general API is used. This holds true e.g. for the basic return codes such as for OK, invalid commands or invalid parameters.*

5.2 Error Codes

TBD

5.3 Type Definitions

The LWM2M module defines several different types for resources. Those are listed in Tab. 15.

Name	*Value*	*Description*	*Size*
BOOL	0x00	Boolean value	4 Byte
INT	0x01	Integer value	4 Byte
FLOAT	0x02	Float value	4 Byte
STRING	0x03	String value	variable
METHOD	0x04	Method	4 Byte

Tab. 15: Different types of a LWM2M resource

5.4 Creation of Objects Instances and Resources

The LWM2M module running on the communication module of the stack provides several levels to define objects and its resources:

- during compile-time (objects/resources are fixed and cannot be deleted)

- during run-time using the according API functions to create objects/instances/resources as described below
- during run-time using an XML file with the according object description as its content. The content shall conform to the IPSO definition standard [2]

5.5 Function Overview

Depending on the firmware configuration of the communication module, some of the methods described above may not exist, e.g. for resource constrained devices (regarding the communication module as well as the host controller) it is feasible to enable the compile-time objects only. Therefore the host controller does not have to create/delete objects but can directly access them via the read and write methods.

Name	*Type*	*Value*	*Direction*	*Description*
LWM2M_CFG_SET	C	0x20	H2D	Set the LWM2M communication parameters. Tab. 18 lists the available configurations.
LWM2M_CFG_GET	C	0x21	H2D	Get the LWM2M communication parameters. Tab. 18 lists the available configurations.
LWM2M_CFG_RET	R	0x22	D2H	Response to the LWM2M_CFG_GET command. Tab. 18 lists the available configurations.
LWM2M_STOP	C	0x30	H2D	Stop the LWM2M communication. The client tries to deregister from the server and stops the LWM2M operation.
LWM2M_START	C	0x31	H2D	Start the LWM2M communication. The client tries to register at the configured server.
LWM2M_BS	C	0x33	H2D	Initiate a bootstrap procedure. The client uses the bootstrap configuration to connect to the bootstrap server.
LWM2M_RESET	C	0x3F	H2D	Reset the LWM2M Layer.
LWM2M_STATUS_GET	C	0x40	H2D	Get the Status of the LWM2M module. This is required for the sensor module to know when the LWM2M communication is ready or if an error occurred.
LWM2M_STATUS _RET	R/U	0x41	D2H	Returns the status of the LWM2M module. The device creates this response either when requested using the LWM2M_STATUS_GET command or automatically if the status of the LWM2M module has changed. Status can be any of the status codes from Tab. 14
LWM2M_ERROR_GET	C	0x50	H2D	Obtain the last error of the LWM2M module.
LWM2M_ERROR_RET	R	0x51	D2H	Returns the latest error of the LWM2M module.
LWM2M_OBJ_CREATE*	C	0x60	H2D	Create a single LWM2M object with a specific instance. The instance can either be given as parameter or can be assigned automatically.
LWM2M_OBJ_CREATE_XML**	C	0x61	H2D	Create a LWM2M object from with a specific instance from an XML file. The instance can either be given as parameter or can be assigned automatically.
LWM2M_OBJ_RET*	R	0x62	D2H	Returns an ID of a previously created L2M2M object. The host uses this ID for further actions related to the according resource (e.g. deletion).
LWM2M_OBJ_DEL*	C	0x63	H2D	Delete a previously created object. The host must use an object ID returned by a previous create operation.
LWM2M_RES_CREATE*	C	0x70	H2D	Create a LWM2M resource and link it to an object and instance.
LWM2M_RES_RET*	R	0x72	D2H	Returns an ID of a previously created L2M2M resource. The host uses this ID for further actions related to the according resource (e.g. deletion).

LWM2M_RES_DEL*	C	0x73	H2D	Delete a previously created resource. The host must use a resource ID returned by a previous create operation.
LWM2M_RES_RD_REQ	U	0x80	D2H	Resource read request initiated by the device. The Device calls this function whenever it receives an according request from the associated LWM2M server.
LWM2M_RES_RD_RSP	R	0x81	H2D	The host has to answer to a LWM2M2_RES_RD_REQ using a LWM2M2_RES_RD_RSP.
LWM2M_RES_WR_REQ	C/U	0x82	BD	Write request to a LWM2M resource. Both device and host use this command e.g. to set configurations or stored values.
LWM2M_RES_WR_RSP	R	0x83	BD	The host/device has to answer to a LWM2M2_RES_WR_REQ using a LWM2M2_RES_WR_RSP.
LWM2M_INST_RD_REQ	U	0x90	D2H	Instance read request initiated by the device. The Device calls this function whenever it receives an according request from the associated LWM2M server.
LWM2M_ INST _RD_RSP	R	0x91	H2D	The host has to answer to a LWM2M2_INST_RD_REQ using a LWM2M2_INST_RD_RSP.
LWM2M_ INST _WR_REQ	C/U	0x92	BD	Write request to a LWM2M instance. Both device and host use this command e.g. to set configurations or stored values.
LWM2M_ INST _WR_RSP	R	0x93	BD	The host/device has to answer to a LWM2M2_INST_WR_REQ using a LWM2M2_INST_WR_RSP.

Tab. 16: Functions of the LWM2M API Set

** Some API functions are available only if the firmware of the communication module provides the creation of objects/instances/resources during runtime.*

***Some API functions are available only if the firmware of the communication module provides the creation of objects/instances/resources during runtime and supports object creation using an XML description file.*

5.6 Function Descriptions

5.6.1 LWM2M_CFG_ (SET/GET_RET)

The configuration functions *LWM2M_CFG_ (SET/GET/RET)* provide the opportunity to change a LWM2M configuration. Each configuration parameter has its default value. The device will use that default value in case the host does not perform a specific configuration. It follows the frame format as shown in Fig. 10.

LEN	*CMD*	*CMD LWM2M*	*Payload*	
			CFG_ID	*CFG_VAL*
3+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 4)	1 Byte	variable

Fig. 10: Frame format of the LWM2M_CFG frame

5.6.1.1 Parameters

Name	*Type*	*Description*
CFG_ID	uint8	ID used to identify the type of configuration to access. The ID can be any of those described in Tab. 18
CFG_VAL	variable	The actual configuration to write, to obtain or indicated.

Tab. 17: Parameters of the LWM2M_CFG frame

5.6.1.2 Configurations

Name	*CFG-ID*	*Type*	*Default Value*	*Description*
BS_SRV_IP	0x00	char[16]	2001::AA	IP address of the LWM2M bootstrap server.*
BS_SRV_PORT	0x01	uint16	5683	Port of the LWM2M bootstrap server.*
SRV_IP	0x02	char[16]	2001::BB	IP address of the LWM2M bootstrap server.*
SRV_PORT	0x03	uint16	5683	Port of the LWM2M bootstrap server.*
CLI_NAME	0x04	string[32]	“emb6lwm2m”	Name of the LWM2M client. The server instance will use this name for identification.*
UPDATE_INTERVAL	0x05	uint32	300	Update Interval of the device.

Tab. 18: LWM2M configuration parameters

**Some configurations can only by applied if the LWM2M layer is in INIT state. In any other state, the device will either reject the request or apply it after the next initialization. Please check the return value of the configuration command to know about the status.*

5.6.2 LWM2M_STOP

The *LWM2M_STOP* command stops a client. Therefore, the client tries to disconnect from the configured server and to de-register its objects.

The command has no parameters and therefore zero length. The *LWM2M_STOP* command does not assume a specific response.

5.6.3 LWM2M_START

The *LWM2M_START* command starts a client. Therefore, the client tries to connect to the configured server and to register its objects.

The command has no parameters and therefore zero length. The *LWM2M_START* command does not assume a specific response.

5.6.4 LWM2M_BS

The *LWM2M_BS* command starts a client initiated bootstrap procedure. Therefore, the client tries to connect to the bootstrap server to obtain information about its server to connect.

The command has no parameters and therefore zero length. The *LWM2M_BS* command does not assume a specific response.

5.6.5 LWM2M_RESET

The *LWM2M_RESET* command resets the LWM2M layer to its initialization state. All resources and settings will be deleted. The LWM2M layer will be stopped.

The command has no parameters and therefore zero length. The *LWM2M_BS* command does not assume a specific response.

5.6.6 LWM2M_STATUS_GET

The *LWM2M_STATUS_GET* command retrieves the status of the device. On reception, the device will respond with its status.

The command has no parameters and therefore zero length. The *LWM2M_STATUS_GET* command expects an *LWM2M_STATUS_RET* response.

5.6.7 LWM2M_STATUS_RET

A device issues a *LWM2M_STATUS_RET* response either on reception of a *LWM2M_STATUS_GET* request or autonomously when its internal status has changed (e.g. if it successfully connected to a server and changed its status to *STATUS_REGISTERED)*. It follows the frame format as shown in Fig. 11.

LEN	*CMD*	*Payload*
		STATUS
2	1 Byte (Tab. 16)	1 Byte

Fig. 11: Frame format of the LWM2M_STATUS response

5.6.7.1 Parameters

Name	*Type*	*Description*
STATUS	ENUMERATION	Status can be any of the status codes from Tab. 14 using a STATUS_* Prefix

Tab. 19: Parameters of the LWM2M_STATUS response

5.6.8 LWM2M_ERROR_GET

A host can use the *LWM2M_ERROR_GET* command to retrieve the specific error code of a device that remains in *LWM2M_STATE_ERROR*.

It has no parameters and therefore zero length. The *LWM2M_ERROR_GET* command expects an *LWM2M_ERROR_RET* response.

5.6.9 LWM2M_ERROR_RET

A device issues an *LWM2M_ERROR_RET* response on reception of an *LWM2M_ERROR_RET* request. It indicates its current specific error when it is in LWM2M_STATE_ERROR. Otherwise, the response returns LWM2M_*ERROR_NO*. It follows the frame format as shown in Fig. 11.

LEN	*CMD*	*Payload*
		ERROR
2	1 Byte (Tab. 16)	1 Byte

Fig. 12: Frame format of the LWM2MERROR response

5.6.9.1 Parameters

Name	*Type*	*Description*
ERROR	ENUMERATION	Error can be any of the error codes from Tab. 14.

Tab. 20: Parameters of the LWM2M ERROR response

5.6.10 LWM2M_OBJ_CREATE

The host uses the *LWM2M_OBJ_CREATE* command to instantiate a new LWM2M object with a specific instance. This command has to be called as long as the device's LWM2M state is initialized but not started yet. Otherwise, the device will reject the command.

To create a LWM2M object the host must send an according object description as a parameter. Therefore, it follows the frame format as shown in Fig. 14. The host expects a *LWM2M_OBJ_RET* response from the device.

LEN	*CMD*	*CMD LWM2M*	*Payload*	
			OBJ_ID	*INST_ID*
4	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte

Fig. 13: Frame format of the LWM2M_OBJ_CREATE_XML frame

5.6.10.1 Parameters

Name	*Type*	*Description*
OBJ_ID	file content	Includes the LWM2M description as an XML file. The file format is specified from OMA and IPSO.
INST_ID	instance ID	Instance ID that shall be used for the object. A value of 0xFF enables automatic generation of the next free instance.

Tab. 21: Parameters of the LWM2M_OBJ_CREATE command

5.6.11 LWM2M_OBJ_CREATE_XML

The host uses the *LWM2M_OBJ_CREATE_XML* command to instantiate a new LWM2M object with a specific instance including its according resources defined in an XML file. This command has to be called as long as the device's LWM2M state is initialized but not started yet. Otherwise, the device will reject the command.

To create a LWM2M object the host must send an according object description as a parameter. Therefore, it follows the frame format as shown in Fig. 14. The host expects a *LWM2M_OBJ_RET* response from the device.

LEN	*CMD*	*CMD LWM2M*	*Payload*	
			INST_ID	*OBJ_DESCR*
3+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	1 Byte	variable

Fig. 14: Frame format of the LWM2M_OBJ_CREATE_XML frame

5.6.11.1 Parameters

Name	*Type*	*Description*
INST_ID	instance ID	Instance ID that shall be used for the object. A value of 0xFF enables automatic generation of the next free instance.
OBJ_DESCR	file content	Includes the LWM2M description as an XML file. The file format is specified from OMA and IPSO.

Tab. 22: Parameters of the LWM2M_OBJ_CREATE command

5.6.12 LWM2M_OBJ_RET

The device sends the *LWM2M_OBJ_RET* response as a reply either to a *LWM2M_OBJ_CREATE, LWM2M_OBJ_CREATE_XML* or a *LWM2M_OBJ_DELETE* command to report the status of the request.

It follows the frame format as shown in Fig. 15.

LEN	*CMD*	*CMD LWM2M*	*Payload*		
			OBJ_ID	*INST_ID*	*STATUS*
6	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	1 Byte

Fig. 15: Frame format of the LWM2M_OBJ_RET frame

5.6.12.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID the response is meant for.
INST_ID	uint8	Instance ID of the object created.
STATUS	ENUMERATION	Status of the operation. Status can be any of the status codes from Tab. 2.

Tab. 23: Parameters of the LWM2M_OBJ_RET frame

5.6.13 LWM2M_OBJ_DEL

The host uses the *LWM2M_OBJ_DEL* command to delete an object from the device LWM2M module.

To delete an object the host has to transmit the according object ID. Therefore, it follows the frame format as shown in Fig. 16. The host expects a *LWM2M_OBJ_RET* response from the device.

LEN	*CMD*	*CMD LWM2M*	*Payload*	
			OBJ_ID	*INST_ID*
5	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte

Fig. 16: Frame format of the LWM2M_OBJ_DEL frame

5.6.13.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID to delete. The host gets the ID during the creation of the object.
INST_ID	uint8	Instance ID of the object to delete. The host gets the ID during the creation of the object.

Tab. 24: Parameters of the LWM2M_OBJ_DEL frame

5.6.14 LWM2M_RES_CREATE

The host uses the LWM2M_RES_CREATE function to create a specific resource at the device LWM2M module. To create a resource the host has to transmit the object and instance IDs the resource belongs to as well as its type. Therefore, it follows the frame format as shown in Fig. 17. The host expects a *LWM2M_RES_RET* response from the device.

LEN	*CMD*	*CMD LWM2M*	*Payload*						
			OBJ_ID	*INST_ID*	*RES_ID*	*TYPE*	*SIZE*	*MOD*	*DATA*
8	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	2 Byte	1Byte	1Byte	1 Byte	Variable (OPTIONAL)

Fig. 17: Frame format of the LWM2M_RES_CREATE frame

5.6.14.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID to the resource belongs to.
INST_ID	uint8	Instance ID the resource belongs to.
RES_ID	uint16	ID of the resource.
TYPE	ENUMERATION	Type of the resource. This can be any type from Tab. 15.
SIZE	uint8	Size of the resource (usually fixed but can vary for type string)
MOD	uint8	Access Modifier
DATA	variable (depends on the resource type)	Initial value of the resource. This field is optional.

Tab. 25: Parameters of the LWM2M_RES_CREATE frame

5.6.15 LWM2M_RES_RET

The device sends the *LWM2M_RES_RET* response as a reply to a *LWM2M_RES_CREATE,* command to report the status of the request. It follows the frame format as shown in Fig. 18.

LEN	*CMD*	*CMD LWM2M*	*Payload*			
			OBJ_ID	*INST_ID*	*RES_ID*	*STATUS*
8	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	2 Byte	1 Byte

Fig. 18: Frame format of the LWM2M_RES_RET frame

5.6.15.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID the response is meant for.
INST_ID	uint8	Instance ID the response is meant for.
RES_ID	uint16	Resource ID that was created.
STATUS	ENUMERATION	Status of the operation. Status can be any of the status codes from Tab. 2.

Tab. 26: Parameters of the LWM2M_RES_RET frame

5.6.16 LWM2M_RES_DEL

The host uses the *LWM2M_RES_DEL* command to delete a resource from the device LWM2M module.

To delete a resource the host has to transmit the according object ID, instance ID and resource ID. Therefore, it follows the frame format as shown in Fig. 19. The host expects a *LWM2M_RES_RET* response from the device.

LEN	*CMD*	*CMD LWM2M*	*Payload*		
			OBJ_ID	*INST_ID*	*RES_ID*
7	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	2 Byte

Fig. 19: Frame format of the LWM2M_RES_DEL frame

5.6.16.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID of the resource to delete. The host gets the ID during the creation of the object.
INST_ID	uint8	Instance ID of the object to delete. The host gets the ID during the creation of the object.
RES_ID	uint8	ID of the resource to delete.

Tab. 27: Parameters of the LWM2M_RES_DEL frame

5.6.17 LWM2M_RES_RD_REQ

The device issues an *LWM2M_RES_RD_REQ* command to read data from an objects resource e.g. upon an according request from the server.

To request a specific value the device has to include the object ID, instance and resource that it wants to access. Therefore, it follows the frame format as shown in Fig. 20. The device expects a *LWM2M_RES_RD_RSP* response from the host.

LEN	*CMD*	*CMD LWM2M*	*Payload*		
			OBJ_ID	*INST_ID*	*RES_ID*
7	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	2 Byte

Fig. 20: Frame format of the LWM2M_RES_RD_REQ frame

5.6.17.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID to read data from. The ID was assigned during the creation of the object.
INST_ID	uint8	Instance ID of the object to read data from. The host gets the ID during the creation of the object.
RES_ID	uint16	Resource ID to read data from.

Tab. 28: Parameters of the LWM2M_RES_RD_REQ frame

5.6.18 LWM2M_RES_RD_RSP

The host uses the *LWM2M_RES_RD_RSP* command to report data to the LWM2M module previously requested by a *LWM2M_RES_RD_REQ* command.

The response includes the according IDs for the Object, Instance and Resource followed by the actual data. The representation of the data depends on the type of the resource that was accessed and therefore its length is variable. It follows the frame format as shown in Fig. 21.

LEN	*CMD*	*CMD LWM2M*	*Payload*			
			OBJ_ID	*INST_ID*	*RES_ID*	*DATA*
7+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	2 Byte	variable

Fig. 21: Frame format of the LWM2M_RES_RD_RSP frame

5.6.18.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID to write the data to. In case the ID is invalid 0xFFFF is returned. Further fields don't have to be considered in that case.
INST_ID	uint8	Instance ID to write the data to. In case the ID is invalid 0xFFFF is returned. Further fields don't have to be considered in that case.
RES_ID	uint16	Resource ID to write the data to. In case the ID is invalid 0xFFFF is returned. Further fields don't have to be considered in that case.
DATA	variable (depends on the resource type)	Actual data to write. The type of the data and its contents depend on the type of the resource that was accessed. E.g. it can be 1 byte length for a uint8 or have variable length in case of a string.

Tab. 29: Parameters of the LWM2M_RES_RD_RSP frame

5.6.19 LWM2M_RES_WR_REQ

The *LWM2M_RES_WR_REQ* command writes data to a specific resource. The command can either be used to update observed values at the device (command is sent by the host) or to update e.g. configurations at the host (command is sent by the device).

The command includes the according IDs for the Object, Instance and Resource followed by the actual data. The representation of the data depends on the type of the resource that was accessed and therefore its length is variable. It follows the frame format as shown in Fig. 22.

LEN	CMD	CMD LWM2M	Payload			
			OBJ_ID	INST_ID	RES_ID	DATA
7+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	2 Byte	variable

Fig. 22: Frame format of the LWM2M_RES_WR_REQ frame

5.6.19.1 Parameters

Name	Type	Description
OBJ_ID	uint16	Object ID the data was read from.
INST_ID	uint8	Instance ID the data was read from.
RES_ID	uint16	Resource ID the data was read from.
DATA	variable (depends on the resource type)	Actual data that was read. The type of the data and its contents depend on the type of the resource that was accessed. E.g. it can be 1 byte length for a uint8 or have variable length in case of a string.

Tab. 30: Parameters of the LWM2M_RES_WR_REQ frame

5.6.20 LWM2M_RES_WR_RSP

The *LWM2M_RES_WR_RSP* is issued from the host or the device as an answer to a previously initiated *LWM2M_RES_WR_REQ* command.

The response only includes the status of the write request. Therefore it follows the frame format as shown in Fig. 23.

LEN	CMD	CMD LWM2M	Payload
			STATUS
7+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	1 Byte

Fig. 23: Frame format of the LWM2M_RES_WR_RSP frame

5.6.21 LWM2M_INST_RD_REQ

The device issues an *LWM2M_INST_RD_REQ* command to read data from an objects resource e.g. upon an according request from the server.

To request a specific value the device has to include the object ID, instance and resources that it wants to access. Therefore, it follows the frame format as shown in Fig. 20. The device expects a *LWM2M_INST_RD_RSP* response from the host.

LEN	CMD	CMD LWM2M	Payload			
			OBJ_ID	INST_ID	NUM	RES_ID[NUM]
6+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	1 Byte	NUM * 2 Byte

Fig. 24: Frame format of the LWM2M_INST_RD_REQ frame

5.6.21.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID to read data from. The ID was assigned during the creation of the object.
INST_ID	uint8	Instance ID of the object to read data from. The host gets the ID during the creation of the object.
NUM	uint8	Number of included resources. If NUM=0 all resources will be requested.
RES_ID[NUM]	uint16	Resource IDs to read data from. This is an array of all the requested resource IDs

Tab. 31: Parameters of the LWM2M_INST_RD_REQ frame

5.6.22 LWM2M_INST_RD_RSP

The host uses the *LWM2M_INST_RD_RSP* command to report data to the LWM2M module previously requested by a *LWM2M_INST_RD_REQ* command.

The response includes the according IDs for the Object, Instance and Resource followed by the actual data. The representation of the data depends on the type of the resource that was accessed and therefore its length is variable. It follows the frame format as shown in Fig. 21.

LEN	*CMD*	*CMD LWM2M*	*Payload*					
			OBJ_ID	*INST_ID*	*NUM*	*RES[NUM]*		
						RES_ID	*SIZE*	*DATA*
6+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	1 Byte	2 Byte	1 Byte	variable

Fig. 25: Frame format of the LWM2M_RES_RD_RSP frame

5.6.22.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID to write the data to. In case the ID is invalid 0xFFFF is returned. Further fields don't have to be considered in that case.
INST_ID	uint8	Instance ID to write the data to. In case the ID is invalid 0xFFFF is returned. Further fields don't have to be considered in that case.
RES	variable (depends on the resource types)	Container for the included resources. Each element consist of the resource ID, the size of the resource and the resource value.
RES_ID	uint16	Resource ID to write the data to. In case the ID is invalid 0xFFFF is returned. Further fields don't have to be considered in that case.
SIZE	uint8	Size of the resource (usually fixed but can vary for type string)
DATA	variable (depends on the resource type)	Actual data to write. The type of the data and its contents depend on the type of the resource that was accessed. E.g. it can be 1 byte length for a uint8 or have variable length in case of a string.

Tab. 32: Parameters of the LWM2M_INST_RD_RSP frame

5.6.23 LWM2M_INST_WR_REQ

The *LWM2M_INST_WR_REQ* command writes data to a specific resource. The command can either be used to update observed values at the device (command is sent by the host) or to update e.g. configurations at the host (command is sent by the device).

The command includes the according IDs for the Object, Instance and the Resources followed by the actual data. The representation of the data depends on the type of the resource that was accessed and therefore its length is variable. It follows the frame format as shown in Fig. 22.

LEN	*CMD*	*CMD LWM2M*	*Payload*					
			OBJ_ID	*INST_ID*	*NUM*	*RES[NUM]*		
						RES_ID	*SIZE*	*DATA*
6+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	2 Byte	1 Byte	1 Byte	2 Byte	1 Byte	variable

Fig. 26: Frame format of the LWM2M_RES_WR_REQ frame

5.6.23.1 Parameters

Name	*Type*	*Description*
OBJ_ID	uint16	Object ID the data was read from.
INST_ID	uint8	Instance ID the data was read from.
RES	variable (depends on the resource types)	Container for the included resources. Each element consist of the resource ID the size of the resource and the resource value.
RES_ID	uint16	Resource ID the data was read from.
SIZE	uint8	Size of the resource (usually fixed but can vary for type string)
DATA	variable (depends on the resource type)	Actual data that was read. The type of the data and its contents depend on the type of the resource that was accessed. E.g. it can be 1 byte length for a uint8 or have variable length in case of a string.

Tab. 33: Parameters of the LWM2M_INST_WR_REQ frame

5.6.24 LWM2M_INST_WR_RSP

The *LWM2M_INST_WR_RSP* is issued from the host or the device as an answer to a previously initiated *LWM2M_INST_WR_REQ* command.

The response only includes the status of the write request. Therefore, it follows the frame format as shown in Fig. 23.

LEN	*CMD*	*CMD LWM2M*	*Payload*
			STATUS
7+x	1 Byte (NEXT_LAYER)	1 Byte (Tab. 16)	1 Byte

Fig. 27: Frame format of the LWM2M_INST_WR_RSP frame

**Functions to access Instances are not implemented yet. Therefore, the according frame formats might still change.*

6 Appendix

6.1 Example Communication Flows

6.1.1 Device Startup

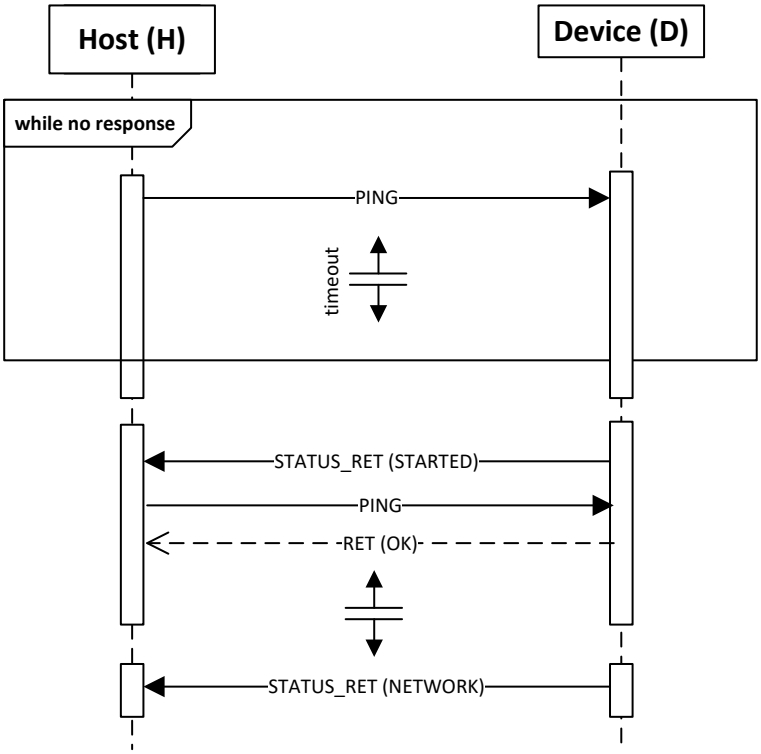

Fig. 28: Sequence: device startup

6.1.2 Configuring a Device

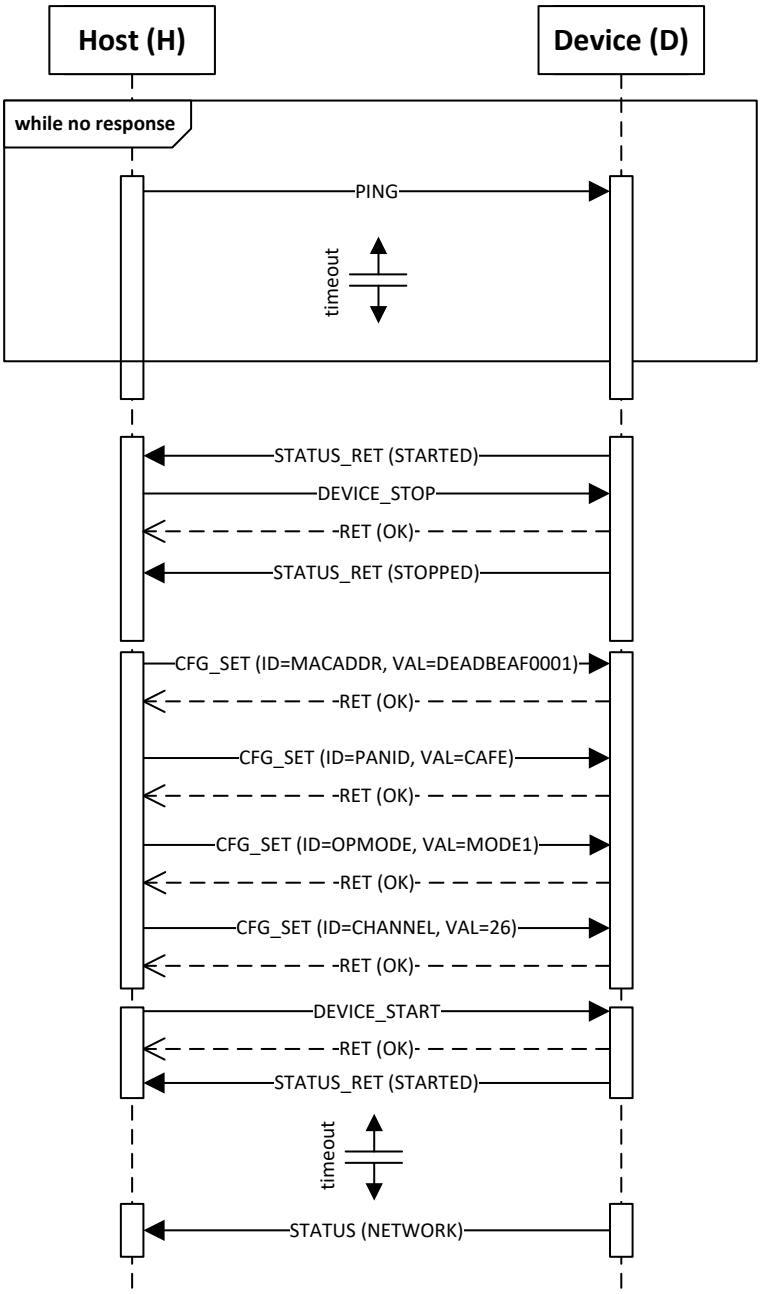

Fig. 29: Sequence: configuring a device

The device starts automatically with its predefined settings. In case the settings shall be changed the device has to be stopped first. After setting the configuration the device has to be started again..

6.1.3 LWM2M Initialization and Configuration (Direct Mode)

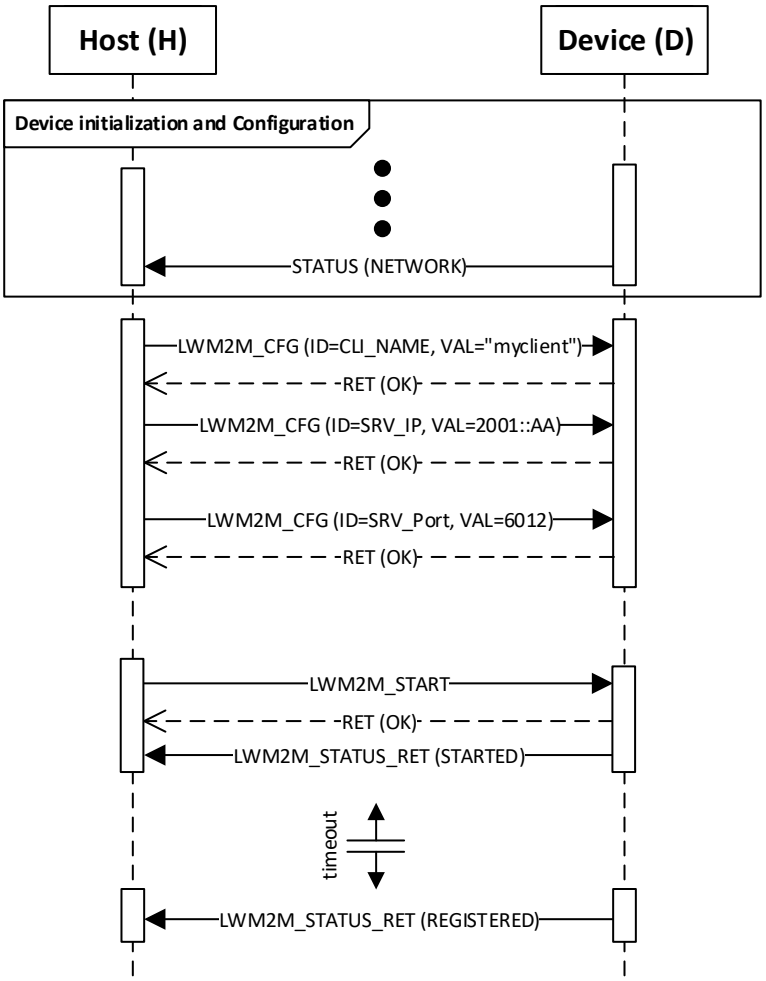

Fig. 30: Sequence: initialization, configuration and start of a LWM2M client in direct mode

After initialization of the device the LWM2M module is stopped by default. It has to be started explicitly from the host after STATUS_NETWORK was reached and after the LWM2M module has been configured accordingly.

6.1.4 LWM2M Initialization and Configuration (Bootstrap Mode)

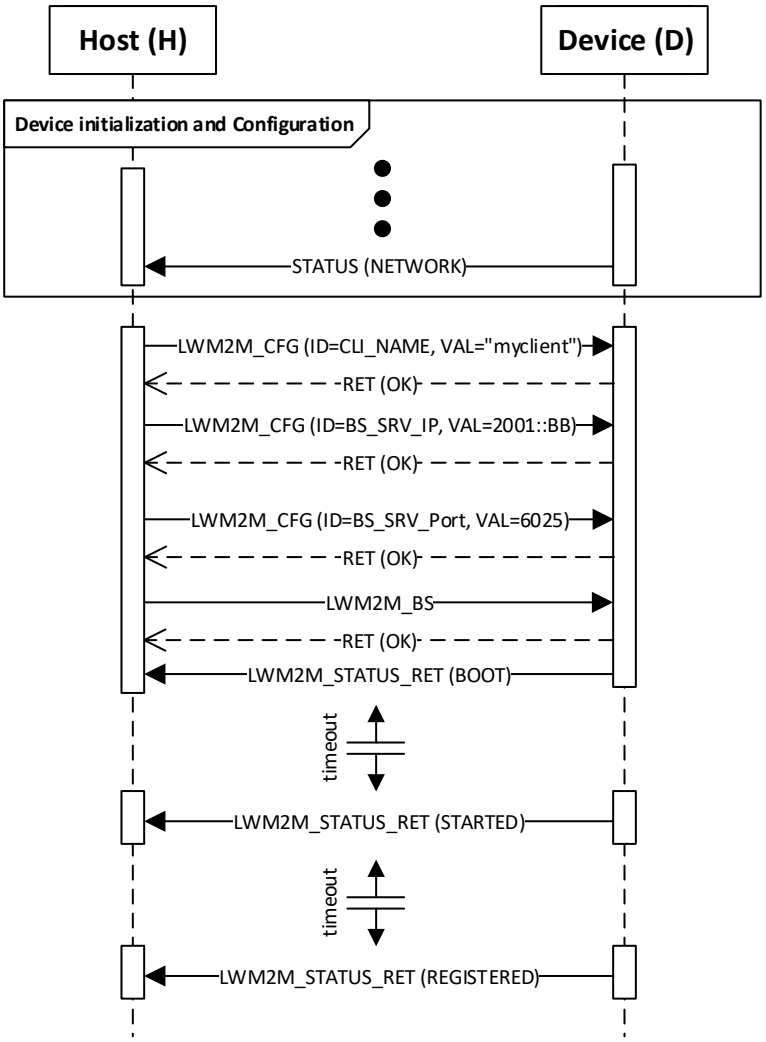

Fig. 31: Sequence: initialization, configuration and start of a LWM2M client in bootstrap mode

After initialization of the device the LWM2M module is stopped by default. Bootstrap It has to be started explicitly from the host after STATUS_NETWORK was reached and after the LWM2M module has been configured accordingly.

6.1.5 LWM2M Resource Creation (Manually)

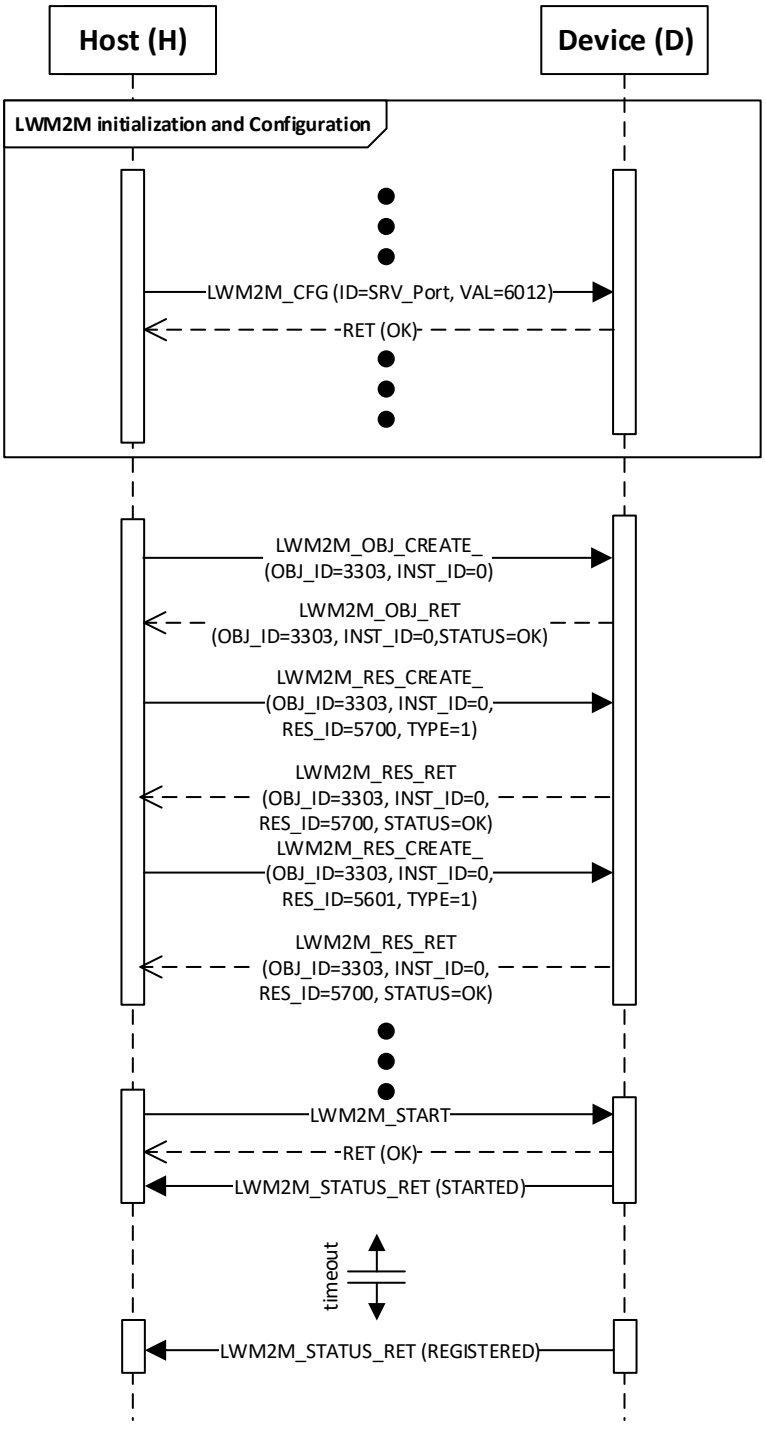

Fig. 32: Sequence: creation of LWM2M resources manually

6.1.6 LWM2M Resource Read Access (Polled Access)

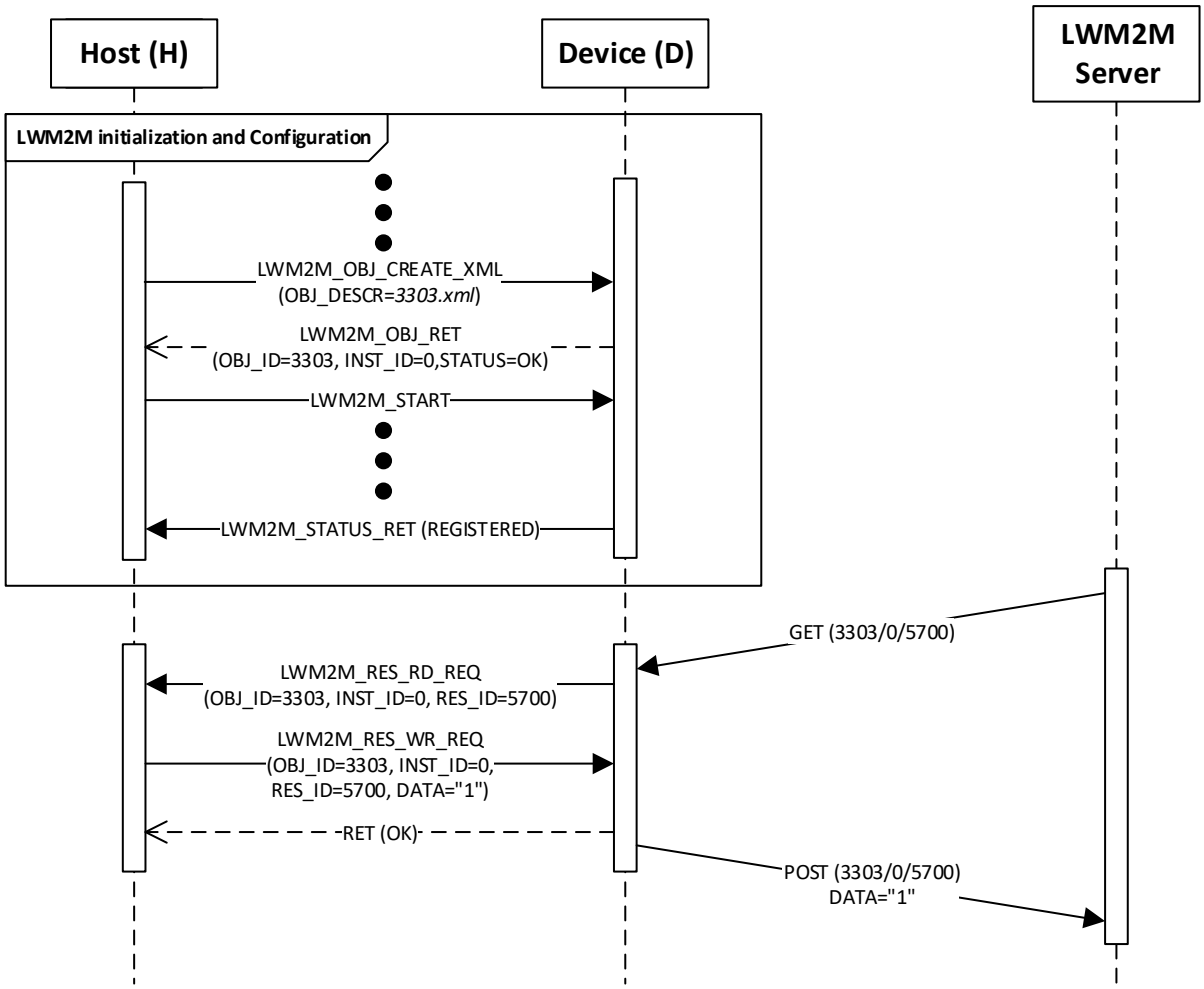

Fig. 33: Sequence: read access of a LWM2M resource

In case static resources are used no objects need to be created. They can be accessed as soon as the client successfully registered at the server. This is indicated by the STATUS_REGISTERED.

6.1.7 LWM2M Resource Read Access (Stored Access)

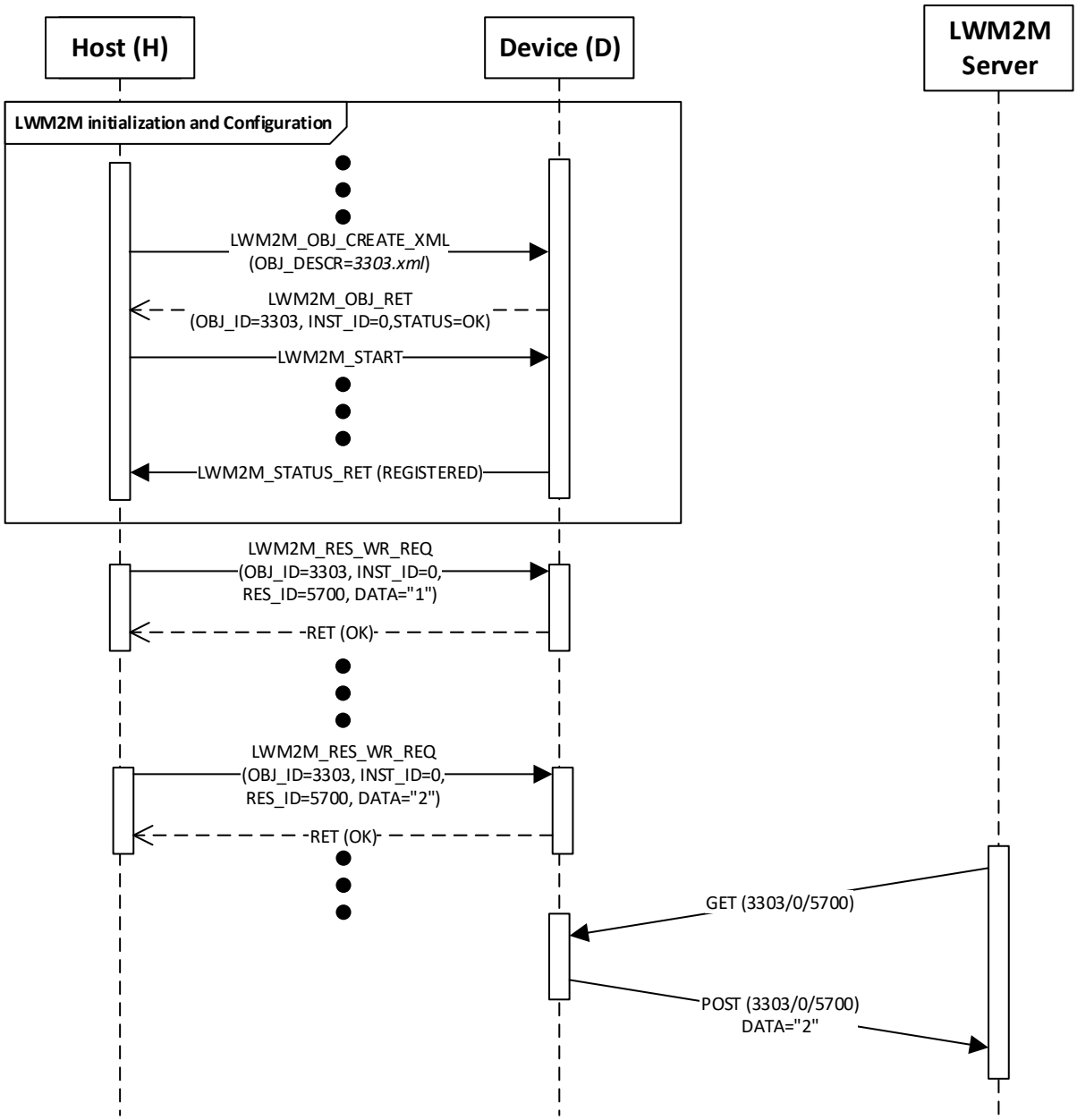

Fig. 34: Sequence: read access of a LWM2M resource

In case static resources are used no objects need to be created. They can be accessed as soon as the client successfully registered at the server. This is indicated by the STATUS_REGISTERED.

6.1.8 LWM2M Resource Write Access

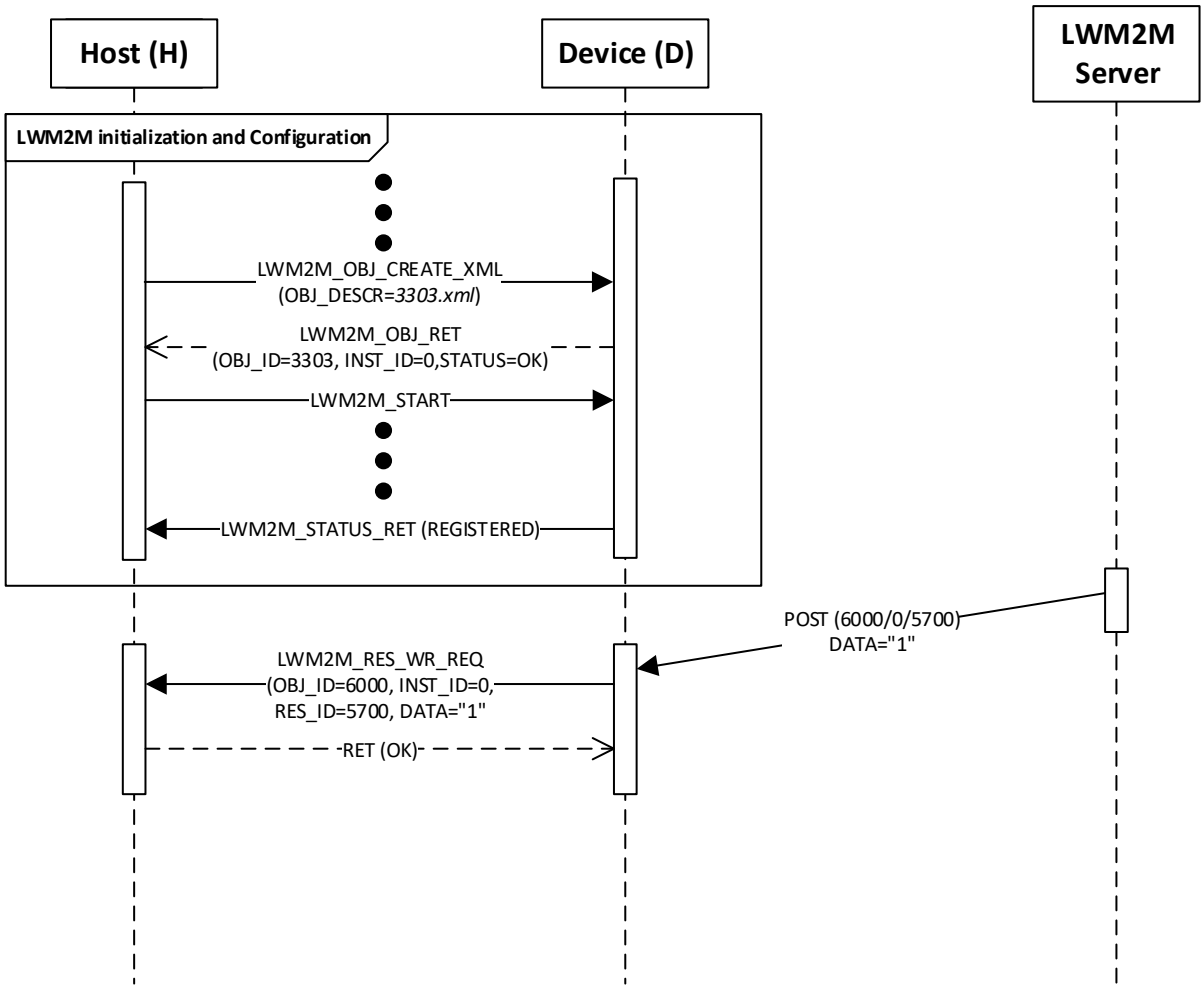

Fig. 35: Sequence: read access of a LWM2M resource

In case static resources are used no objects need to be created. They can be accessed as soon as the client successfully registered at the server. This is indicated by the STATUS_REGISTERED.

6.1.9 LWM2M Resource Observe Access

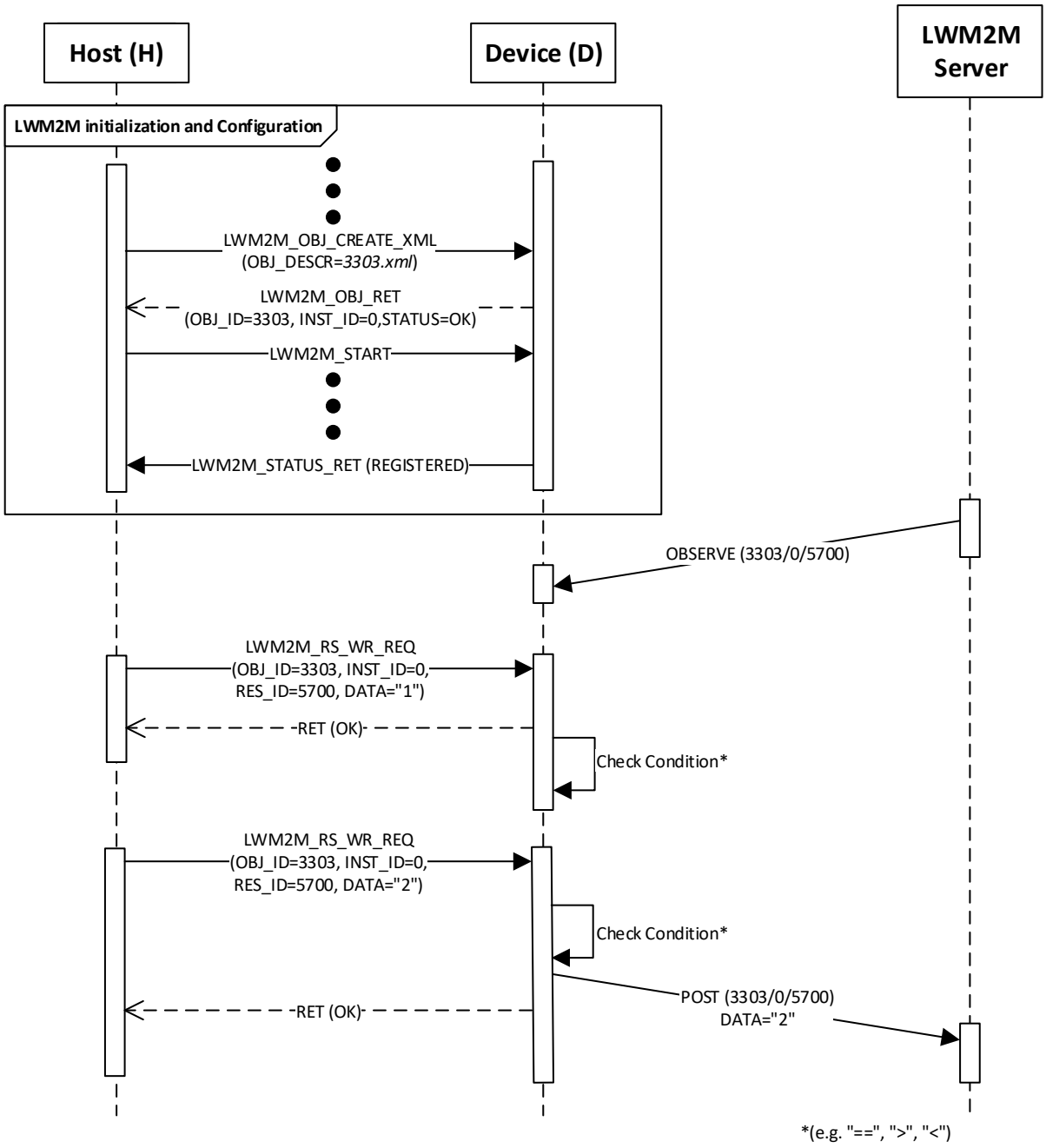

Fig. 36: Sequence: observe access of a LWM2M resource

In case static resources are used no objects need to be created. They can be accessed as soon as the client successfully registered at the server. This is indicated by the STATUS_REGISTERED.

6.2 LWM2M Temperature Sensor Description

```
<?xml version="1.0" encoding="utf-8"?>
<LWM2M>
```

```
   <Object ObjectType="MODefinition">
      <Name>Temperature</Name>
      <Description1>Description: This IPSO object should be used with a temperature sensor to
report a temperature measurement.  It also provides resources for minimum/maximum measured values
and the minimum/maximum range that can be measured by the temperature sensor. An example
measurement unit is degrees Celsius (ucum:Cel).</Description1>
      <ObjectID>3303</ObjectID>
      <ObjectURN>urn:oma:lwm2m:ext:3303</ObjectURN>
      <MultipleInstances>Multiple</MultipleInstances>
      <Mandatory>Optional</Mandatory>
      <Resources>
         <Item ID="5700">
            <Name>Sensor Value</Name>
            <Operations>R</Operations>
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Mandatory</Mandatory>
            <Type>Float</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units>Defined by “Units” resource.</Units>
            <Description>Last or Current Measured Value from the Sensor</Description>
         </Item>
         <Item ID="5601">
            <Name>Min Measured Value</Name>
            <Operations>R</Operations>
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Optional</Mandatory>
            <Type>Float</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units>Defined by “Units” resource.</Units>
            <Description>
             The minimum value measured by the sensor since power ON or reset
           </Description>
         </Item>
         <Item ID="5602">
            <Name>Max Measured Value</Name>
            <Operations>R</Operations>
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Optional</Mandatory>
            <Type>Float</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units>Defined by “Units” resource.</Units>
            <Description>
             The maximum value measured by the sensor since power ON or reset
            </Description>
         </Item>
         <Item ID="5603">
            <Name>Min Range Value</Name>
            <Operations>R</Operations>
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Optional</Mandatory>
            <Type>Float</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units>Defined by “Units” resource.</Units>
            <Description>The minimum value that can be measured by the sensor</Description>
         </Item>
         <Item ID="5604">
            <Name>Max Range Value</Name>
            <Operations>E</Operations>
```

```
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Optional</Mandatory>
            <Type>Float</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units>Defined by "Units" resource.</Units>
            <Description>The maximum value that can be measured by the sensor</Description>
         </Item>
         <Item ID="5701">
            <Name>Sensor Units</Name>
            <Operations>R</Operations>
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Optional</Mandatory>
            <Type>String</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units></Units>
            <Description>
              Measurement Units Definition e.g. "Cel" for Temperature in Celsius.
           </Description>
         </Item>
         <Item ID="5605">
            <Name>Reset Min and Max Measured Values</Name>
            <Operations>E</Operations>
            <MultipleInstances>Single</MultipleInstances>
            <Mandatory>Optional</Mandatory>
            <Type>String</Type>
            <RangeEnumeration></RangeEnumeration>
            <Units></Units>
            <Description>Reset the Min and Max Measured Values to Current Value</Description>
         </Item>
      </Resources>
      <Description2></Description2>
   </Object>
</LWM2M>
```

List. 2: LWM2M description of a temperature sensor

References

[1] “emb::6 specification”, Institute of Reliable Embedded Systems and Communication Electronics (ivESK)

[2] “IP for Smart Objects - IPSO Objects”, Public Github of the IPSO Alliance, https://github.com/IPSO-Alliance/pub