LWM2M Guide

Cross-Domain Development Kit XDK110
Platform for Application Development

Bosch Connected Devices and Solutions

BOSCH
Invented for life

XDK110: Data Sheet

Document revision	1.4
Document release date	06/03/2017
Document number	BCDS-XDK110-GUIDE-LWM2M
Technical reference code(s)	
Notes	Data in this document is subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance. **Subject to change without notice**

XDK LWM2M Guide

Platform for Application Development

In times of the Internet-of-Things it becomes increasingly important to find an appropriate way to connect different devices. For now most of the current internet protocols are meant for complex use cases, like browsing the internet with a laptop, but when it comes to more specialized devices like the XDK these protocols are oversized, often. The LWM2M protocol is a new approach to solve this issue. This guide shows the basic functionality of how to use the XDK LWM2M API which provides all components that are needed to configure a LWM2M client on the XDK.

Table of Contents

This guide postulates a basic understanding of the XDK and according Workspace. For new users we recommend going through the following guides at xdk.io/guides first:

- *Workbench Installation*
- *Workbench First Steps*
- *XDK Guide FreeRTOS*
- *XDK Guide Wi-Fi*

1. Introduction

1.1 OMA LWM2M

OMA Lightweight M2M is a protocol from the Open Mobile Alliance for M2M communication or Internet-of-Things device management that is set on top of the Constrained Application Protocol[1]. It was specified to reduce the unnecessary overhead created by other protocols like *http* when connecting simple, embedded devices.[2]

The fundamental idea of LWM2M is to provide a predefined structure so that the clients only have to send the actual data. Besides the fact that this reduces the transmitted data considerably, it also helps to standardize the interfaces so every object implementing the protocol can be used the same way. Of course this only comes with certain limitations concerning flexibility, but on the other hand this makes not only a client, but also a server very lightweight, since only one model have to be implemented.

The data model basically is a container for a consecutively numbered amount of objects, wich again have consecutively numbered resources. An **object** covers a certain use case whilst its **resources** provide more details in information and functionality. For now OMA is reserving different scopes for different uses[3]:

Table 1. Object ID Classes

Category	Object ID Range	Description
`oma-label`	0 - 1023	Objects defined by the Open Mobile Alliance
`reserved`	1024 - 2047	Reserved for future use
`ext-label`	2048 - 10240	Objects defined by a 3rd party SDO
`x-label`	10241 - 32768	Objects defined by a vendor or individual such an object may be either private (no DDF or Specification made available) or public. These objects are optionally private this is indicated at the time to submission

The higher numbers are open to custom usage, which comes in handy if the functionality, that is needed to model, is not there, yet. For a more detailed list of yet used object IDs see: http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry

This guide will concentrate on the **Object-ID 3** which represents the **Device** itself. It has 17 resources that split up as follows.

[1] For more information about CoAP see http://coap.technology/

[2] cf. http://www.eclipse.org/leshan/ „WHAT IS LWM2M?“

[3] cf. http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry

Table 2. Device Object Resources

Id	Name	Type	Operations	Multiple Instances	Mandatory
0	Manufacturer	String	Read	Single	Optional
1	Model Number	String	Read	Single	Optional
2	Serial Number	String	Read	Single	Optional
3	Firmware Version	String	Read	Single	Optional
4	Reboot	-	Execute	Single	Mandatory
5	Factory Reset	-	Execute	Single	Optional
6	Available Power	Integer	Read	Multiple	Optional
7	Power Source Voltage	Integer	Read	Multiple	Optional
8	Power Source Current	Integer	Read	Multiple	Optional
9	Battery Level	Integer	Read	Single	Optional
10	Memory Free	Integer	Read	Single	Optional
11	Error Code	Integer	Read	Multiple	Mandatory
12	Reset Error Code	-	Execute	Single	Optional
13	Current Time	Time	Read, Write	Single	Optional
14	UTC Offset	String	Read, Write	Single	Optional
15	Timezone	String	Read, Write	Single	Optional
16	Supported Binding and Modes	String	Read	Single	Mandatory

This Table shows a brief overview of the resources the object contains. For the full information including the description of the object and all properties of any resource see the specification on: http://technical.openmobilealliance.org/tech/profiles/LWM2M_Device-v1_0.xml

Note: There are mandatory objects like the LWM2M Security, Server and Device objects and mandatory resources within different objects. In order to prevent errors the not-implemented mandatory objects are handled internally as far as possible.

1.2 Project Leshan

Basically, the Leshan project is a server-client implementation of the LWM2M protocol in Java. Since the XDK uses rather C than Java, only the server component is needed. Therefore the Leshan project even has a standalone LWM2M server everyone can freely connect to:

The connection takes places via CoAP: coap://leshan.eclipse.org
The clients can be seen on: http://leshan.eclipse.org/#/clients

Note: The Leshan sandbox is open to everyone, so be careful with the information and functionality you provide. Since it is just a playground for testing the client, custom objects are not supported, unfortunately.

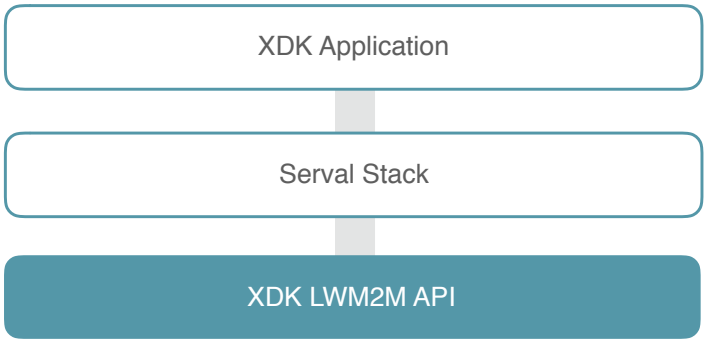
2. LWM2M on XDK

In the following chapters the reader will get familiar with the LWM2M API on the XDK and even develop an own small client. After that the reader will be able to expand the client according to own purposes.

2.1 LWM2M API

This chapter introduces the basics of the LWM2M API by providing condensed information of the most important features. To see the full content, the API can be consulted in the workbench[4] or on the XDK website[5].

Picture 1. API Hierarchy



The API is situated in the serval stack. For full API access the following header bus be included at the top of the *implementation file (.c)*:

Code 1: Including LWM2M Interface

```
#include <Serval_Lwm2m.h>
```

Table 3. LWM2M Interface

Interface	Description
`Serval_Lwm2m.h`	Interface to the implementation of the lightweight M2M protocol specified by the Open Mobile Alliance for machine to machine communication

Note: The specification of the lightweight M2M protocol has not been finalized, yet. Therefore, the current implementation may change to meet possible changes in the specification!

[4] open any project in the workbench and navigate to:
`./SDK/xdk110/Libraries/Serval/ServalStack/api/Serval_Lwm2m.h`

[5] http://xdk.bosch-connectivity.com/xdk_docs/html/_serval___lwm2m_8h.html

The whole interface can be controlled by the following functions:

Table 4. Sample of LWM2M API functions

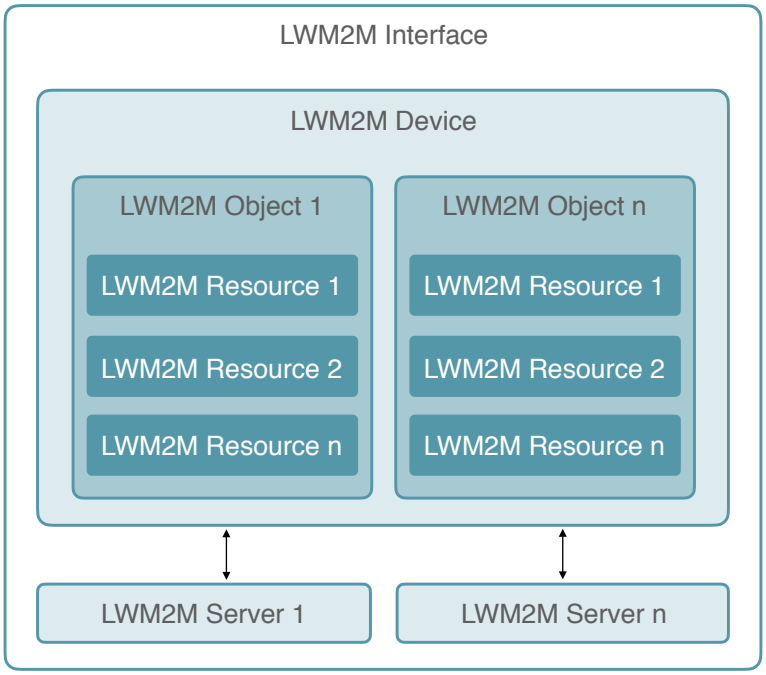
Function	Description
`Lwm2mRegistration_register()`	Register a LwM2M device on a certain LWM2M server
`Lwm2mRegistration_registerToAllServers()`	Register a LwM2M device to all specified LWM2M servers
`Lwm2mRegistration_update()`	Perform a registration update of a LWM2M device on a certain LWM2M server
`Lwm2mRegistration_updateToAllServers()`	Perform a registration update of a LWM2M device on all specified LWM2M servers
`Lwm2mRegistration_deRegister()`	Perform a deregistration of a LWM2M device on a certain LWM2M server
`Lwm2mRegistration_deRegisterFromAllServers()`	Perform a deregistration of a LWM2M device on all specified LWM2M servers
`Lwm2m_getServer()`	Get a reference to a particular server
`Lwm2m_setNumberOfServers()`	Sets the number of instanciated servers. This function should be called after the application set the server information
`Lwm2m_initialize()`	Initialize the Lightweight M2M module
`Lwm2m_start()`	Start the Lightweight M2M module

2.2 API usage

In this chapter a basic LWM2M client that connects to the Eclipse Leshan Project sandbox server, will be set up. Afterwards the XDK can be found on http://leshan.eclipse.org/#/clients.

The structure of the implementation can be imagined roughly as follows:

Picture 2. LWM2M client implementation model



The LWM2M API is providing data objects for the different LWM2M items. These objects are acting as containers for the next smaller objects. The interface allows to declare one device instance that represents the XDK. Within this device object multiple LWM2M objects can be defined which again can contain multiple LWM2M resources. The device can connect to different servers.

2.2.1 Preparation

This implementation is based on the `XdkApplicationTemplate`. The template can be opened by clicking on *Help* > *Welcome* > *XDKApplicationTemplate* in the workbench. By default the application starts in the `appInitSystem()` function which should already be at the end of the implementation file. First, all required interfaces need to be included, in addition to the ones that are already there:

Code 2. Additional required interfaces

```
// LWM2M interface
#include <Serval_Lwm2m.h>

// WiFi settings
#include "BCDS_NetworkConfig.h"
#include "BCDS_WlanConnect.h"

// socket settings
#include "PAL_initialize_ih.h"
#include "PAL_socketMonitor_ih.h"
```

Before any client can be implemented, the XDK has to be connected to the internet. The code below shows a very basic example of a function for a connection via WPA. For different connection methods or custom settings, please consult the Wi-Fi guide mentioned at the beginning.

After connecting to the internet, a socket can be intialized, that is needed for the LWM2M communication:

Code 3. Preparing network connection

```
void appInitSystem(xTimerHandle xTimer)
{
    (void) (xTimer);

    WlanConnect_SSID_T connectSSID = (WlanConnect_SSID_T) "networkSSID";
    WlanConnect_PassPhrase_T connectPassPhrase = (WlanConnect_PassPhrase_T)
"PW";

    WlanConnect_Init();
    NetworkConfig_SetIpDhcp(0);
    WlanConnect_WPA(connectSSID, connectPassPhrase, NULL);

    PAL_initialize();
    PAL_socketMonitorInit();
}
```

Note: `appInitSystem()` need to be left at the end of the implementation file.

2.2.2 Defining LWM2M device resources

This paragraph shows how to configure the devices objects and resources.
First the device need to be configured by using the `Lwm2mDevice_T` type to define the general properties of the XDK:

Table 5. Lwm2mDevice_T properties

Property	Description
`.name`	The name that will be used as the clients name by the LWM2M server.
`.binding`	The binding function the XDK will choose. Possible values are: *LWM2M_BINDING_UNDEFINED* *LWM2M_BINDING_QUEUED* *UDP* *SMS* *UDP_QUEUED* *SMS_QUEUED* *UDP_AND_SMS* *UDP_QUEUED_AND_SMS* *LWM2M_BINDING_QUEUED* For information on the possible values habe a look at the type definition of `Lwm2m_Binding_T`.
`.sms`	Telephone number of the device used during communication via SMS (NULL for not available)
`.numberOfObjectInstances`	The number of object instances contained in the device

Property	Description
`.objectInstances`	The list of object instances of the device. This list will only contain application specific objects and the device object

The object instances list itself can be described by the type `Lwm2mObjectInstance_T`:

Table 6. Lwm2mObjectInstance_T properties

Property	Description
`.objectId`	The LWM2M object identifier
`.objectInstanceId`	The instance id of the instance
`.resources`	Pointer to the resources associated with this instance
`.maxNumberOfResources`	The max number of resources for this object
`.permissions`	Array of permission sorted by server (first permission refers to the first server in the server array) This will be used to synthetise the Access Control List objects

To define the resources the `LWM2M_RESOURCES(resources)` helper macro could be used, which will count and set the resources that are configured:

Code 4. Defining and setting device information

```
struct DeviceResource_S
{
    Lwm2mResource_T manufacturer;
};

struct DeviceResource_S deviceResources =
{
    { 0, LWM2M_STRING_RO( "Bosch Connected Devices and Solutions GmbH" ) },
};

Lwm2mObjectInstance_T objectInstances[] =
{
    {
        // object ID:
        3,
        // instance ID:
        0,
        // set resources:
        LWM2M_RESOURCES(deviceResources),
        // set permissions:
        .permissions = {0x3F, 0x0, 0x0, 0x0}
    },
};

Lwm2mDevice_T deviceResourceInfo =
{
    .name = "YOUR_DEVICE_NAME",
    .binding = UDP,
    .sms = NULL,
    .numberOfObjectInstances = 1,
    .objectInstances = objectInstances,
};
```

This is the most basic setup to start the client. Only the manufacturer is specified in this example. In comparison with *Table 2* it stands out that this implementation of the resources is missing the mandatory objects. In the context of this guide this will suffice, since this is just a demonstration of how to declare a resource in general. The missing resources will be intercepted, so there are still results to see on the Leshan Server.

The following code example will show how to add more resources. Once there is a basic understanding it will be easy to expand a definition. Please do not copy the following code in the application right now, because it is just for showing the schema. The same principle as for adding a new resource applies to adding new instances and objects:

Code 5. Adding more resources

```
struct DeviceResource_S
{
    Lwm2mResource_T manufacturer;
    Lwm2mResource_T yourSecondResource;
};

struct DeviceResource_S deviceResources =
{
    { 0, LWM2M_STRING_RO( "Bosch Connected Devices and Solutions GmbH" ) },
    { 1, LWM2M_STRING_RO( "second Resource" ) },
};
```

The name in the `DeviceResource_S` struct is just a placeholder to keep the resources apart of each other.

Note: If the implementation doesn't stick to the LWM2M definition of an object it might lead to an inappropriate behavior on the server. So, for example the Device Object expects resource two to be the Model Number. If a date object would be passed here, useless information might be displayed on the server.

For setting the resources data types correctly the following helper macros can be used:

Table 7. Sample of LWM2M API helper macros

Macro	Description
`LWM2M_STRING_RO(string)`	Helper macro to initialize a string resource
`LWM2M_INTEGER(integer)`	Helper macro to initialize an integer resource
`LWM2M_FLOAT(floating)`	Helper macro to initialize a floating point resource
`LWM2M_BOOL(boolean)`	Helper macro to initialize a bool resource
`LWM2M_TIME(time)`	Helper macro to initialize a time resource
`LWM2M_DYNAMIC(dyn)`	Helper macro to initialize a dynamic resource
`LWM2M_DYNAMIC_ARRAY(dyn)`	Helper macro to initialize multiple instance resources
`LWM2M_FUNCTION(function)`	Helper macro to initialize a function resource

2.2.3 Initializing the LWM2M Interface

Now the device itself and it's objects and resources are defined, the interface can be initialized with the settings that are previously made:

Code 6. Initialize LWM2M Client

```
// put this at the end of the appInitSystem() after the socket init
Lwm2m_initialize(&deviceResourceInfo);
```

2.2.4 Configuring the LWM2M Server

Before starting the client it is necessary to tell the client to which servers it should connect to. By default the LWM2M implementation of the XDK allows up to four servers:

Table 8. Maximum number of servers

Macro	Value
LWM2M_MAX_NUMBER_SERVERS	4

If more than one server instance is needed, the following steps have do be done for each instance:

Code 7. Server settings

```
// put this at the end of the appInitSystem() after the LWM2M init

char* serverAddress = "coap://5.39.83.206:5683";

// get the first server instance
Lwm2mServer_T* server = Lwm2m_getServer(0);
strncpy(server->serverAddress, serverAddress, strlen(serverAddress));
server->permissions[0] = LWM2M_READ_ALLOWED;
Lwm2m_setNumberOfServers(1);
```

Note: The server instance wants to have an IP address, not an URL. The IP above is one IP from the Leshan sandbox[6].

In this code example the server permissions were set to LWM2M_READ_ALLOWED[7]. See the following table to learn which alternatives exist:

[6] Effective date 19.06.2016. If there is trouble with the IP just ping leshan.eclipse.org

[7] This includes the rights for reading and observing attributes as well as to write attributes that are specified writable by the LWM2M protocol

Table 9. Sample of LWM2M API access right macros

Macro
LWM2M_READ_ALLOWED
LWM2M_WRITE_ALLOWED
LWM2M_EXECUTE_ALLOWED
LWM2M_DELETE_ALLOWED
LWM2M_CREATE_ALLOWED
LWM2M_FULL_ACCESS

2.2.5 Starting the LWM2M Client

The LWM2M interface requires a callback function that can tell the application about events. Possible events are:

Table 10. LWM2M API event types

Macro	Description
LWM2M_EVENT_TYPE_BOOTSTRAP	Bootstrapping
LWM2M_EVENT_TYPE_REGISTRATION	Registration
LWM2M_EVENT_TYPE_REGISTRATION_UPDATE	Registration Update
LWM2M_EVENT_TYPE_DEREGISTRATION	Deregistration
LWM2M_EVENT_TYPE_WRITE	Write operation executed
LWM2M_EVENT_TYPE_OBJECT_CREATED	Create operation executed
LWM2M_EVENT_TYPE_OBJECT_DELETED	Delete operation executed
LWM2M_EVENT_TYPE_NEW_OBSERVER	Observe operation executed
LWM2M_EVENT_TYPE_NOTIFICATION	Notification operation executed
LWM2M_EVENT_TYPE_OBSERVATION_CANCELED	Cancel operation executed
LWM2M_EVENT_TYPE_NEW_SERVER_ADDED	New Server was added

The application callback needs to be implemented as follows:

Code 8. Application callback function

```
static void applicationCallback(Lwm2m_Event_Type_T eventType, Lwm2m_URI_Path_T
*path, Retcode_T status)
{
    // do something
}
```

With this callback function the start command can be executed:

Code 9. Start the client

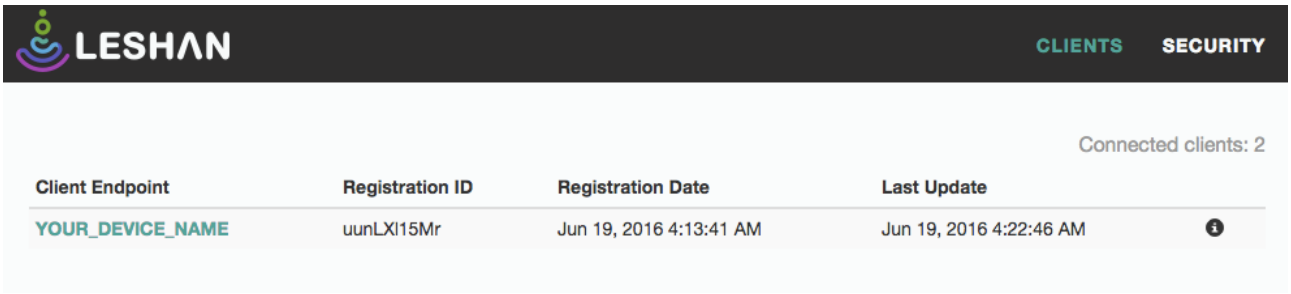
```
// put this at the end of the appInitSystem() after the LWM2M server settings

// choose any free port
Ip_Port_T port = Ip_convertIntToPort(1234);

Lwm2m_start(port, &applicationCallback);
// register at first server instance
Lwm2mRegistration_register(0);
```

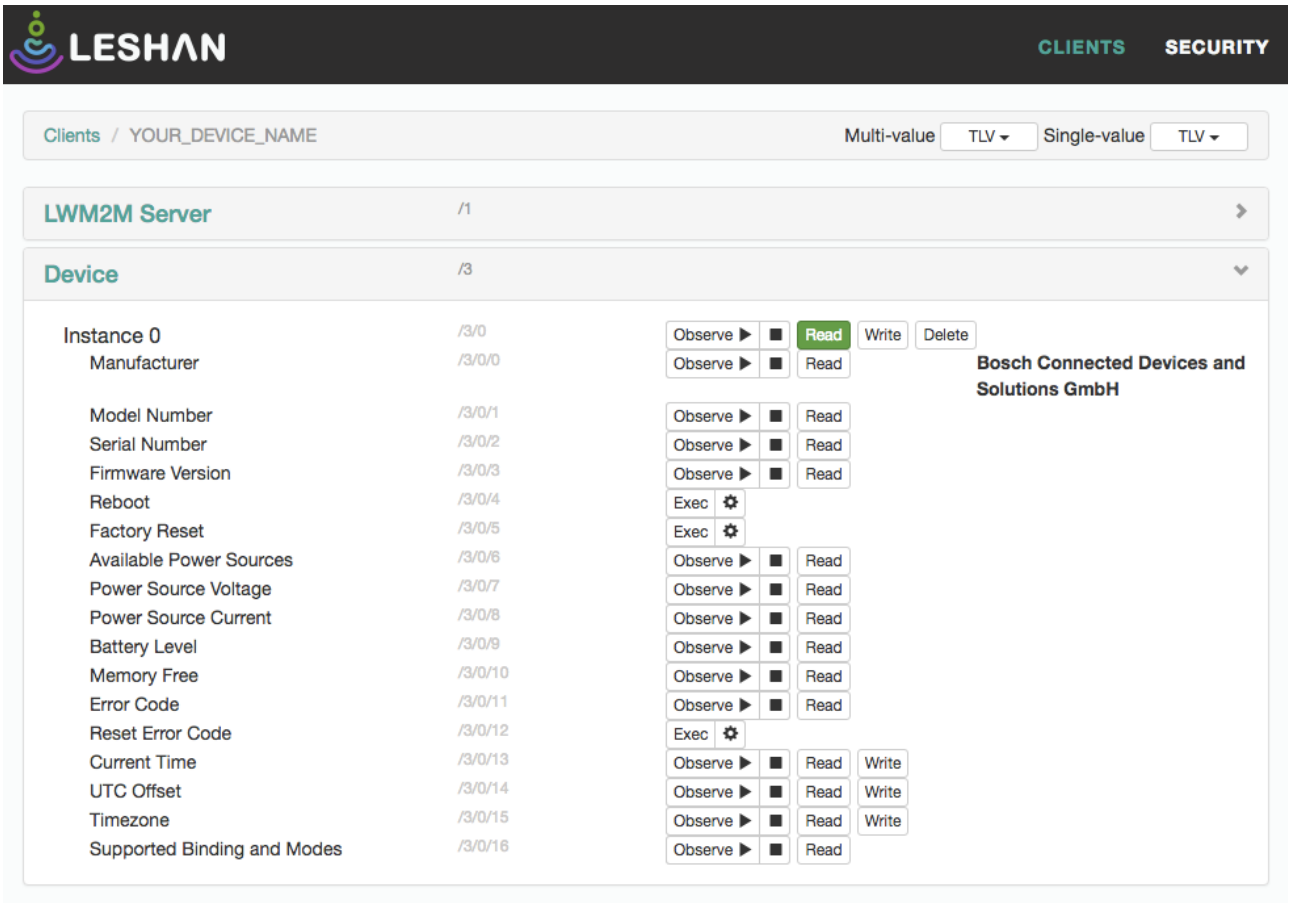
If the application is flashed to the XDK now, the client should start properly. After the application is started the client will be shown on the Eclipse Leshan site http://leshan.eclipse.org/#/clients in the client list:

Picture 3. Leshan client list



On the device detail page the property that was defined before is shown:

Picture 4. Leshan client details



2.2.6 Keeping the LWM2M Client running

After a while it will happen that „*read*“ can’t be executed on the client any more because of a timeout error which results of the fact that the registration was only sent once and has never been updated. To keep the client running, an implementation of a re-registration routine is needed. Therefore a timer will be created that must refresh the registration from time to time:

Code 10. Registration Update

```
void RegistrationUpdate(xTimerHandle pxTimer)
{
    (void) pxTimer;

    // update registration for server at index 0
    Lwm2mRegistration_update(0);
}
```

Code 11. Registration Update Timer

```
// put this at the end of the appInitSystem() after the LWM2M registration

// Variables for the timer task
uint32_t timerAutoReloadOn = UINT32_C(1);
uint32_t twentySecondsDelay = UINT32_C(20000) / portTICK_RATE_MS;

xTimerHandle registrationUpdateTimerHandler =
        xTimerCreate((const char * const ) "RegistrationUpdate",
        twentySecondsDelay,
        timerAutoReloadOn,
        (void *) NULL,
        RegistrationUpdate
        );

xTimerStart(registrationUpdateTimerHandler, 0);
```

2.2.7 Error handling

As many of the XDK API interfaces the LWM2M functions are also having a return code. In this guide the return codes have been ignored for the purpose of understanding. Below an example of how to evaluate the return code is shown. It is recommended to handle every API call like this:

Code 12. Return code example

```
// replace the previous RegistrationUpdate() with this:

void RegistrationUpdate(xTimerHandle pxTimer){

    (void) pxTimer;

    Retcode_T rc = RC_OK;

    rc = Lwm2mRegistration_update(0);

    if (RC_OK != rc){
        printf("send registration update failed: %i \r\n", rc);
        return;
    }
    else {
        printf("registration update was sent... \r\n");
    }
}
```