# Eureka LwM2M Proxy Sample Project

Nordic Semiconductor ASA

Version 0.2 (released 2019-12-05)

## System Overall

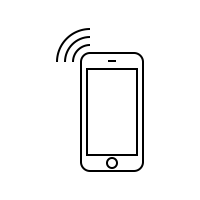
The system setup is like below.

PCA10090 DK



nRF52840

nRF9160



Cellular

BLE

nRF52840 / nF52832

PCA10056 / PCA10040 DK

On the PCA10090 DK hardware, nRF52840\_PCA10090 is the board controller and route some GPIOs from nRF9160 to Arduino pins. nRF52840\_PCA10056 act as the main MCU, for BLE communication towards smart devices and for sensor data collecting.

By connecting the Arduino GPIO pins exposed by nRF9160 to nRF52840\_PCA10056 Arduino pins, we create a UART connection between them.

The same applies to nRF52832\_PCA10040 as the external and main MCU.

A protocol is prepared for data traffic over UART, which support

* Arbitrary data traffic
* Modem control service
* LwM2M service

nRF9160 side is under full control by nRF52840, including LTE connection and cloud access.

The start-up sequence is defined as below:

1. nRF91 always start up and enter sleep;
2. nRF52 start up and start a periodical timer to toggle an interface GPIO
3. nRF52 de-assert an interface GPIO
4. Once nRF91 is woke up, send a SYNC command to nRF52
5. On receiving the SYNC command, nRF52 acknowledge and proceed

The reference code uses below configuration for the interface GPIO.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **nRF52840** | | **nRF9160** | | **Comment** |
| **OUT** | P0.27 | **IN** | P0.31 | Active low, Sense enabled on nRF91 |

After wake-up, the GPIO connection could be used for other purpose, for example reset UART connection when error occurs.

This feature requires CONFIG\_THIN\_GPIO\_WAKEUP to be set. Default is yes.

## Hardware Setup

PCA10090 has default setting of VDD IO to be 1.8V, needs to be adjusted to 3.0V by SW11. Check [this link](https://infocenter.nordicsemi.com/topic/ug_nrf91_dk/UG/nrf91_DK/hw_debug_out.html?cp=2_0_2_4_8_0). **NOTE this will cause issue for LTE connection**. A level-shift in between might be the best solution.

Also, in this setup, on PCA10090 side we choose to use the GPIO pins that is NOT under the routing by on-board nRF52840, thus need to be careful in selecting exactly the pins to use.

GPIO pin assignment for UART interface:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **nRF52840/nRF52832** | | **nRF9160** | | **Comment** |
| **TX** | P0.6 | **RX** | P0.11 | Assigned in project DTS overlay file |
| **RX** | P0.8 | **TX** | P0.10 | Assigned in project DTS overlay file |
| **CTS** | P0.7 | **RTS** | P0.12 | Assigned in project DTS overlay file |
| **RTS** | P0.5 | **CTS** | P0.13 | Assigned in project DTS overlay file |

UART instance used: nRF52840/nRF52832 (UART0) nRF9160 (UART2)

Interface configuration:

Hardware flow control: enabled

Baud-rate: 115200

Parity bit: no

Operation mode: IRQ (DMA mode to be studied)

UART0 on nRF9160 sides are used for logging, while UART1 on nRF9160 side is for VCOM by default. nRF52840 can also choose to use UART1, while nRF52832 can only use UART0.

## Protocol over UART

A simple protocol is prepared to provide services of nRF9160 side to nRF52 side. The protocol packets are transported on UART interface between nRF91 and nRF52.

The AT command service is the basic and fundamental service, which is always enabled. Other services are listed (but not limited) below:

* Modem control service
* MQTT Client service
* CoAP Client service (planned)
* LWM2M Client service
* SNTP Client service
* GPS service
* TCP/IP TCP and UDP Client service

This protocol supports max 8 services, and max 16 commands for each service.

NOTE max size of payload his protocol supported by this protocol is 255 bytes.

## 3.1 Protocol Packet Format

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STX (0x02) | Type  (one-byte) |  | Length  (one-byte) | Value  (vary) | BCC  (one-byte) |

Type – Data, Command, Response, Notification

Length – Length of Value, max 255 bytes

BCC – Block Check Code, XOR from Type to Value

BCC calculation:

*BCC = 0x00;*

*len = sizeof(control\_protocol);*

*for(i=1; i<(len-2); i++){*

*BCC = BCC ^ control\_protocol[i];*

*}*

## 3.2 Protocol definition

This is a symmetric protocol, with two logical roles, transmitter and receiver.

The MSB bit in Type field depicts the traffic direction

‘0 : transmitter to receiver, e.g. Data, Control Command

‘1: Receiver to transmitter, e.g. Command response, unsolicited notification

Every solicited control command Type should have one corresponding response Type, with only difference on MSB bit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **I/O** | **Type** | **Length** | **Value** | **Comment** |
| TX/RX | 0x00 | Max 255 | Hex data | Special type for bi-directional arbitrary data, no ACK |
| TX | 0x01 | 0 | 0 | SYNC command |
| TX | 0x02 | 0 | 0 | SLEEP command |
|  | | | | |
| TX | 0x10 | 0 | 0 | Modem control, init modem and connect network |
| TX | 0x11 | 0 | 0 | Modem control, go offline |
| TX | 0x12 | 0 | 0 | Modem control, power off |
| TX | 0x13 | 0 | 0 | Modem control, go online |
| TX | 0x14 | 1 | PSM mode | Modem control, set PSM 0: disabled, 1: enabled |
| TX | 0x15 | 1 | eDRX mode | Modem control, set eDRX 0: disabled, 1: enabled |
|  | | | | |
| TX | 0x20 | 2 | Lifetime | Connect to Bootstrap or LwM2M server |
| TX | 0x21 | 0 | 0 | Disconnect from LwM2M server |
| TX | 0x22 | Various | Path string | Set resource URI for read/write |
| TX | 0x23 | Various | Read integer | Read int from resource URI |
| TX | 0x24 | Various | Write integer | Write int to resource URI |
| TX | 0x25 | Various | Read float | Read float from resource URI |
| TX | 0x26 | Various | Write float | Write float to resource URI |
| TX | 0x27 | Various | Read string | Read string from resource URI |
| TX | 0x28 | Various | Write string | Write string to resource URI |
| TX | 0x29 | Various | Read opaque | Read opaque data from resource URI |
| TX | 0x2A | Various | Write opaque | Write opaque data to resource URI |
|  | | | | |
| TX | 0x7F | Reserved | Reserved | Reserved |
|  | | | | |
| RX | 0x80 | Reserved | Reserved | Reserved |
| RX | 0x81~0xFE | Various | responses | Command responses |
|  | | | | |
| TX/RX | 0xFF | Variable | Notification data | Special type for unsolicited notifications |

**Command Response**

|  |  |
| --- | --- |
| **Command Result**  **(one byte)** | **Command Response Data**  **(various bytes)** |
| 0x00 | Command executed, other response data follows |
| <error code> | Command execution error, code is defined in [1]  If there is more information, followed by response data |

**Unsolicited Notification**

|  |  |
| --- | --- |
| **Notification Type**  **(one byte)** | **Notification Data**  **(various bytes)** |
| 0x00 | Reserved |
| 0x01 | Unsolicited LwM2M RD event message, refer to 3.2.4 |
| 0x02 | Unsolicited LwM2M Object event message, refer to 3.2.4 |

### 3.2.1 Modem Control Service

This service is always enabled. Below flags are added to “eureka\_lwm2m\_proxy\prj.conf”

*# LTE link control*

*CONFIG\_LTE\_AUTO\_INIT\_AND\_CONNECT=n*

*CONFIG\_LTE\_NETWORK\_MODE\_LTE\_M=y*

Please NOTE LTE\_NETWORK\_MODE could be adjusted accordingly. More details on other configuration items in “nrf\drivers\lte\_link\_control”.

### 3.2.4 LwM2M service

**LwM2M Connect**

MQTT client can connect to a LwM2M peer server with an endpoint name. Depends on below flags, the peer server is used in two different ways.

CASE#1 Register to LwM2M Server directly

*CONFIG\_LWM2M\_PEER\_PORT=5683*

*CONFIG\_LWM2M\_RD\_CLIENT\_SUPPORT\_BOOTSTRAP=n*

CASE#2 Register to Bootstrapping Server first

*CONFIG\_LWM2M\_PEER\_PORT=5783*

*CONFIG\_LWM2M\_SERVER\_INSTANCE\_COUNT=2*

*CONFIG\_LWM2M\_RD\_CLIENT\_SUPPORT\_BOOTSTRAP=y*

Customer always need to specify LwM2M or Bootstrap server in “prj.conf”. It could either domain name or IPv4 address.

*config ELP\_LWM2M\_SERVER*

*string "LwM2M server"*

*help*

*"hostname or IP address of LwM2M remote server"*

When nRF52 issue LwM2M Connect, it needs to specify registration lifetime in seconds, minimal 15, and maximal 65535. If invalid, default 180s applies. For example,

inter\_connect\_send(CMD\_TYPE\_LWM2M\_CONNECT, (uint8\_t \*)&lifetime, 2);

The proxy uses “i:<IMEI>” as the default LwM2M client name.

**LwM2M Disconnect**

LwM2M Disconnect enable client to disconnect from a connected LwM2M server. When nRF52 issue LwM2M Disconnect, it needs not to specify any parameter. For example,

inter\_connect\_sendCMD\_TYPE\_LWM2M\_DISCONNECT, NULL, 0);

**LwM2M Set Path**

LwM2M Set Path enable client to set the resource URI for next read/write operation. This command requires parameters of “*[path(var)]*”. The path is standard URI of Lwm2M resource model, e.g. “03311/0/5850”.

**LwM2M Read Integer**

LwM2M Read Integer enable client to read integer type of data from pre-set resource URI. This command requires parameters of “*[integer type(1)]*”, while *integer type* is defined as

*enum lwm2m\_int\_type {*

*LWM2M\_INT\_TYPE\_BOOLEAN, // Length 1 byte*

*LWM2M\_INT\_TYPE\_UINT8, // Length 1 byte*

*LWM2M\_INT\_TYPE\_UINT16, // Length 2 byte*

*LWM2M\_INT\_TYPE\_UINT32, // Length 4 byte*

*LWM2M\_INT\_TYPE\_UINT64, // Length 1 byte*

*LWM2M\_INT\_TYPE\_INT8, // Length 1 byte*

*LWM2M\_INT\_TYPE\_INT16, // Length 2 byte*

*LWM2M\_INT\_TYPE\_INT32, // Length 4 byte*

*LWM2M\_INT\_TYPE\_INT64 // Length 8 byte*

*};*

The response for this command is “*[result][val(1~8)]*”. In case of success, *result* is 0. Otherwise it’s a negative error value. The *val* is the integer read out from the resource. NOTE the *val* is coded in little-endian format.

**LwM2M Write Integer**

LwM2M Write Integer enable client to write integer type of data to pre-set resource URI. This command requires parameters of “*[integer type(1)][value(1~8)]*”. The *integer type* is defined as above. The *value* must be coded in little-endian format. The length of *value* is dependent on type. The response for this command is “*[error(1)]*”, while 0 means success, otherwise a negative error code.

**LwM2M Read Float**

LwM2M Read Float enable client to read float type of data from a given resource URI. This command requires parameters of “*[float type(1)]*”, while *float type* is defined as

*enum lwm2m\_float\_type {*

*LWM2M\_FLOAD\_TYPE\_32, // 8 bytes*

*LWM2M\_FLOAD\_TYPE\_64 // 16 bytes*

*};*

NOTE LwM2M use fixed-point float, which is defined as below:

*/\*\**

*\* @brief Data structure used to represent the LwM2M float type:*

*\* val1 is the whole number portion of the decimal*

*\* val2 is the decimal portion \*1000000 for 32bit, \*1000000000 for 64bit*

*\* Example: 123.456 == val1: 123, val2:456000*

*\* Example: 123.000456 = val1: 123, val2:456*

*\*/*

*/\*\**

*\* @brief 32-bit variant of the LwM2M float structure*

*\*/*

*typedef struct float32\_value {*

*s32\_t val1;*

*s32\_t val2;*

*} float32\_value\_t;*

*/\*\**

*\* @brief 32-bit variant of the LwM2M float structure*

*\*/*

*typedef struct float64\_value {*

*s64\_t val1;*

*s64\_t val2;*

*} float64\_value\_t;*

The response for this command is “*[result(1)[val1(4 or 8)] [val2(4 or 8)]*”. In case of success, *result* is 0. Otherwise it’s a negative error value. *Val1/2* is the float read out from the resource. NOTE the *Val1/2* is coded in little-endian format.

**LwM2M Write Float**

LwM2M Write float enable client to write float type of data to pre-set resource URI. This command requires parameters of “*[float type(1)][val1(4 0r 8)] [val2(4 0r 8)]*”. The *float type* is defined as above. The *val1/2* must be coded in little-endian format. The length of *value* is dependent on type. The response for this command is “*[error(1)]*”, while 0 means success, otherwise a negative error code.

**LwM2M Read String**

LwM2M Read String enable client to read string type of data from pre-set resource URI.

The response for this command is “*[result(1)][val(var)]*”. In case of success, *result* is 0. Otherwise it’s a negative error value. *Val* is the string value read out from the resource.

**LwM2M Write String**

LwM2M Write String enable client to write string type of data to pre-set resource URI. This command requires parameters of “*[value(var)]*”. The response for this command is “*[error(1)]*”, while 0 means success, otherwise a negative error code.

**LwM2M Read Opaque**

LwM2M Read Opaque enable client to read opaque type of data from pre-set resource URI.

The response for this command is “*[result(1)][val(var)]*”. In case of success, *result* is 0. Otherwise it’s a negative error value. *Val* is the opaque value read out from the resource.

**LwM2M Write Opaque**

LwM2M Write Opaque enable client to write opaque type of data to pre-set resource URI. This command requires parameters of “*[value(var)]*”. The response for this command is “*[error(1)]*”, while 0 means success, otherwise a negative error code.

**LwM2M Unsolicited Event**

There are three types of Unsolicited events to be sent to nRF52

/\*\* Unsolicited notification type. \*/

*enum ic\_notify\_type {*

*NOT\_TYPE\_BASE = 0x00,*

*/\*\* Notification type LwM2M \*/*

*NOT\_TYPE\_LWM2M\_RD, /\* Registration and discovery events \*/*

*NOT\_TYPE\_LWM2M\_OBJECT, /\* LwM2M Object events \*/*

*NOT\_TYPE\_LWM2M\_NOTIFY\_RESULT, /\* LwM2M Notify result events \*/*

*NOT\_TYPE\_INVALID*

*};*

The RD events are defined as below:

*enum lwm2m\_rd\_client\_event {*

*LWM2M\_RD\_CLIENT\_EVENT\_NONE,*

*LWM2M\_RD\_CLIENT\_EVENT\_BOOTSTRAP\_REG\_FAILURE,*

*LWM2M\_RD\_CLIENT\_EVENT\_BOOTSTRAP\_REG\_COMPLETE,*

*LWM2M\_RD\_CLIENT\_EVENT\_BOOTSTRAP\_TRANSFER\_COMPLETE,*

*LWM2M\_RD\_CLIENT\_EVENT\_REGISTRATION\_FAILURE,*

*LWM2M\_RD\_CLIENT\_EVENT\_REGISTRATION\_COMPLETE,*

*LWM2M\_RD\_CLIENT\_EVENT\_REG\_UPDATE\_FAILURE,*

*LWM2M\_RD\_CLIENT\_EVENT\_REG\_UPDATE\_COMPLETE,*

*LWM2M\_RD\_CLIENT\_EVENT\_DEREGISTER\_FAILURE,*

*LWM2M\_RD\_CLIENT\_EVENT\_DISCONNECT*

*};*

The Object events is specific to Object resource definition.

## Implementation

The software module structure is described as below:

nRF52 sample projects

nRF5\_SDK\_15.3.0\examples\peripheral\**lwm2m\_client** target: nrf52832 / nrf52840

nRF9160 serial-lte-modem

<NCS>\nrf\samples\nrf9160\**eureka\_lwm2m\_proxy** target: nrf9160\_pca10090ns

Please use a terminal to get nRF91 log by opening the largest number of <COMx> (exposed by PCA10090) and use RTT Viewer to get nRF52 log from PCA10040/PCA10056.

When *CONFIG\_THIN\_TEST* is defined and *CONFIG\_THIN\_GPIO\_WAKEUP* is undefined, local test functions would be called.

## Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Comments** |
| 2019/11/29 | 0.1 | Initial draft release |
| 2019/12/5 | 0.2 | LwM2M set path, read/write opaque data |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Reference

[1] Zephyr LIBC Error Code in GitHub, [link](https://github.com/NordicPlayground/fw-nrfconnect-zephyr/blob/master/lib/libc/minimal/include/errno.h)

[2] MQTT Publish and Subscribe Beginners Guide, [link](http://www.steves-internet-guide.com/mqtt-publish-subscribe/)

[3] GPS NMEA data format, [link](https://www.gpsinformation.org/dale/nmea.htm)