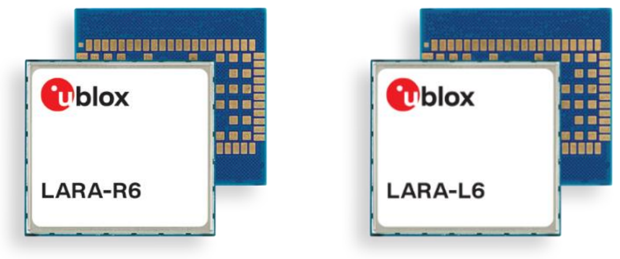
C

# LARA-R6 / LARA-L6 series

## Linux integration

Application note



**Abstract**

This document describes the integration of LARA-R6 and LARA-L6 series modules on Linux OS platforms with RmNet and CDC-ECM technologies.

UBX-22026570 - R03
C1-Public

# Document information

| | | |
|---|---|---|
| **Title** | **LARA-R6 / LARA-L6 series** | |
| **Subtitle** | Linux integration | |
| **Document type** | Application note | |
| **Document number** | UBX-22026570 | |
| **Revision and date** | R03 | 29-Nov-2023 |
| **Disclosure restriction** | C1-Public | |

This document applies to the following products:

| Product series | Notes |
|---|---|
| LARA-R6 | Except "00B" product version |
| LARA-L6 | |

# Contents

# 1 Introduction

LARA-R6 / LARA-L6 series modules support 2G, 3G, and LTE radio access technology (RAT). Regardless of the selected RAT, the packet switched connectivity over the USB interface may be established in two different networking modes:

- **RmNet** mode
- **ECM** mode

This document describes the two operating modes, how the IP connectivity is provided, and the Linux commands to establish IP connectivity for the application at DTE level control.

In this document, the u-blox cellular module is referred to as data communication equipment (DCE), or as the “target”. The device connected to the module is referred to as data terminal equipment (DTE). The physical connection between the target and the DTE is established through the high-speed USB 2.0 interface, which support different logical interfaces (ACM, DIAG, RMNET, ECM).

## 1.1 RmNet

Qualcomm mobile station modem (MSM) Interface (QMI) defines the DCE-DTE interface, which exposes various functionalities of MSM, including tethered networking.

RmNet interface is a new logical device in QMI framework for data services. It defines channels for control and data transfers. For example, it uses data I/O channel for IP data transfer and control I/O channel for QMI messaging, which is similar to AT commands. RmNet may replace legacy USB modem interface.

On the host side system, the RmNet device (LARA-R6 / LARA-L6) appears as an ethernet adapter without ethernet framing/MAC support.

## 1.2 CDC-ECM

Communications device class (CDC) ethernet control mode (ECM) is a USB class that implements ethernet emulation on a USB device. Packet switched connectivity over the USB virtual ethernet interface may be established in two different networking modes:

- **Bridge mode:** the IP termination of the data connectivity is on the customer’s application processor. The module acts as a bridge device.
- **Router mode:** the IP termination is on the module. The data connectivity of the customer’s application processor is provided through routing procedures. The module acts as a mobile router.

# 2 Linux DTE integration

## 2.1 Operating System setup

### 2.1.1 Summary

LARA-L6 series modules expose different usb-device according to the Product ID.

| No. | VID/PID | Device type | Kernel driver/module | Description |
|---|---|---|---|---|
| 1 | 0x1546/0x1341 or 0x1342 ox 0x1343 | Four serial ports (reduced CDC-ACM) | option | Standard module port for non-IP usage (AT commands, NMEA and DIAGnostic) functionalities |
| 2 | 0x1546/0x1343 | Network adapter following CDC-ECM standard | cdc _ether | Provides Router-mode capability within a Private IP addressing scenario |
| 3 | 0x1546/0x1342 | RmNet mobile broadband adapter | qmi_wwan, cdc_wdm | Provides multiple-PDN connection and best speed performances. It also provides the cdc-wdm control interface to the DTE. |
| 4 | 0x1546/0x1341 or 0x1342 ox 0x1343 | QHSUSB__BULK Single serial port (reduced CDC-ACM) | option | Special module condition when DUMP protocol can be used (ex. post firmware crash) |
| 5 | 0x05C6/0x9008 | QDL mode | qcserial | Flashing using Qualcomm VID/PID |

**Table 1 List of USB devices exposed by LARA-L6 series modules**

LARA-R6 series modules expose different usb-device according to the Product ID.

| No. | VID/PID | Device type | Kernel driver/module | Description |
|---|---|---|---|---|
| 1 | 0x1546/0x1311 or 0x1312 ox 0x1313 | Four serial ports (reduced CDC-ACM) | option | Standard module port for non-IP usage (AT commands, NMEA and DIAGnostic) functionalities |
| 2 | 0x1546/0x1313 | Network adapter following CDC-ECM standard | cdc _ether | Provides Router-mode capability within a Private IP addressing scenario |
| 3 | 0x1546/0x1312 | RmNet mobile broadband adapter | qmi_wwan, cdc_wdm | Provides multiple-PDN connection and best speed performances. It also provides the cdc-wdm control interface to the DTE. |
| 4 | 0x1546/0x1311 or 0x1312 ox 0x1313 | QHSUSB__BULK Single serial port (reduced CDC-ACM) | option | Special module condition when DUMP protocol can be used (ex. post firmware crash) |
| 5 | 0x05C6/0x9008 | QDL mode | qcserial | Flashing using Qualcomm VID/PID |

**Table 2 List of USB devices exposed by LARA-R6 series modules**

☞ Kernel patches for option, qmi_wwan and cdc_ether have been accepted to the Linux kernel. With the latest kernel version, Linux can automatically enumerate LARA-R6 / LARA-L6 module.

### 2.1.2 Driver tailoring

Standard free open-source Linux drivers are used. **Kernel v.5.12.x** or later are recommended.

Older kernel must be patched or updated to properly use RmNet and to get multiple-PDN supports for Qualcomm multiplexing and aggregation protocol (QMAP).

To support LARA-R6 / LARA-L6 USB VID and PID and the serial-ports, add the following lines to `./drivers/usb/serial/option.c`, in kernel

```
#define UBLOX_VENDOR_ID                         0x1546
#define UBLOX_PRODUCT_LARA_L6                   0x1341
#define UBLOX_PRODUCT_LARA_L6_RMNET             0x1342
#define UBLOX_PRODUCT_LARA_L6_ECM        0x1343
#define UBLOX_PRODUCT_LARA_R6                   0x1311
#define UBLOX_PRODUCT_LARA_R6_RMNET             0x1312
#define UBLOX_PRODUCT_LARA_R6_ECM        0x1313

…
{ USB_DEVICE(UBLOX_VENDOR_ID, UBLOX_PRODUCT_LARA_R6) },
{ USB_DEVICE(UBLOX_VENDOR_ID, UBLOX_PRODUCT_LARA_R6_RMNET),
        .driver_info = RSVD(4) },
{ USB_DEVICE_INTERFACE_CLASS(UBLOX_VENDOR_ID, UBLOX_PRODUCT_LARA_R6_ECM, 0xff) },
{ USB_DEVICE(UBLOX_VENDOR_ID, UBLOX_PRODUCT_LARA_L6),
      .driver_info = RSVD(4) },
{ USB_DEVICE(UBLOX_VENDOR_ID, UBLOX_PRODUCT_LARA_L6_RMNET),
      .driver_info = RSVD(4) },
{ USB_DEVICE(UBLOX_VENDOR_ID, UBLOX_PRODUCT_LARA_L6_ECM),
      .driver_info = RSVD(4) },
```

To support RmNet mode, add the following lines to `./drivers/net/usb/qmi_wwan.c` in kernel

```
{QMI_QUIRK_SET_DTR(0x1546, 0x1342, 4)},
{QMI_QUIRK_SET_DTR(0x1546, 0x1312, 4)},
```

To support ECM mode, add the following lines to `./drivers/net/usb/cdc_ether.c` in kernel

```
…
{
 USB_DEVICE_AND_INTERFACE_INFO(UBLOX_VENDOR_ID, 0x1313, USB_CLASS_COMM,
                          USB_CDC_SUBCLASS_ETHERNET,
                          USB_CDC_PROTO_NONE),
 .driver_info = (unsigned long)&wwan_info,
}, {
 USB_DEVICE_AND_INTERFACE_INFO(UBLOX_VENDOR_ID, 0x1343, USB_CLASS_COMM,
                          USB_CDC_SUBCLASS_ETHERNET,
                          USB_CDC_PROTO_NONE),
 .driver_info = (unsigned long)&wwan_info,
},
…
```

Use proper kernel build-system tools, (cross-)compiler and packages to replace original ones.

```
# cd linux-source-5.15.0
# make -C /lib/modules/$(uname -r)/build  M=$(pwd)/drivers/usb/serial clean
# make -C /lib/modules/$(uname -r)/build  M=$(pwd)/drivers/net/usb clean

# make -C /lib/modules/$(uname -r)/build  M=$(pwd)/drivers/usb/serial
# cp -v drivers/usb/serial/option.ko /lib/modules/$(uname -r)/kernel/drivers/usb/serial

# make -C /lib/modules/$(uname -r)/build  M=$(pwd)/drivers/net/usb
# cp -v drivers/net/usb/qmi_wwan.ko  /lib/modules/$(uname -r)/kernel/drivers/net/usb
# cp -v drivers/net/usb/cdc_ether.ko /lib/modules/$(uname -r)/kernel/drivers/net/usb
```

### 2.1.3 Driver loading

If not properly loaded by `udev`, the driver modules could be force-loaded at the next boot by adding them in `/etc/modules`.

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the name of kernel modules that shall be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
option
qmi_wwan
cdc_ether
```

To manually check if the drivers are loaded:

```
# modprobe -v option
# modprobe -v qmi_wwan
# modprobe -v cdc_ether

# lsmod | grep option
option         61440   0
usb_wwan       24576   1 option
usbserial      57344   2 usb_wwan,option

# lsmod | grep qmi_wwan
qmi_wwan       36864  0
cdc_wdm               28642 1       qmi_wwan
usbnet         53248  2       qmi_wwan      ,cdc_ether

# lsmod | grep cdc_ether
cdc_ether      24576  0
usbnet         53248  2       qmi_wwan,cdc_ether
```

### 2.1.4 Module connection

Power on the LARA-R6 / LARA-L6 module connected to the Linux DTE by USB 2.0.

Check if the module has been detected (in the examples below LARA-L6 is used):

```
# lsusb
…
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 007: ID 1546:134x U-Blox AG u-blox Modem
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
…
```

### 2.1.5 USB Device driver enumeration

The following checking can be done if the module has been properly detected.

```
# lsusb -t
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 5000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
    |__ Port 2: Dev 2, If 0, Class=Hub, Driver=hub/3p, 480M
        |__ Port 1: Dev 6, If 1, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 1: Dev 6, If 0, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 2: Dev 4, If 0, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 3: Dev 5, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
    |__ Port 3: Dev 7, If 0, Class=Vendor Specific Class, Driver=option, 480M
    |__ Port 3: Dev 7, If 1, Class=Vendor Specific Class, Driver=option, 480M
    |__ Port 3: Dev 7, If 2, Class=Vendor Specific Class, Driver=option, 480M
    |__ Port 3: Dev 7, If 3, Class=Vendor Specific Class, Driver=option, 480M
    |__ Port 3: Dev 7, If 4, Class=Vendor Specific Class, Driver=qmi_wwan, 480M
    |__ Port 8: Dev 3, If 1, Class=Wireless, Driver=btusb, 12M
    |__ Port 8: Dev 3, If 0, Class=Wireless, Driver=btusb, 12M
```

Kernel provides the following standard debug messages inside a classic `dmesg` logging system.

```
…
[ 2040.456032] usb 1-3: new high-speed USB device number 7 using xhci_hcd
[ 2040.609791] usb 1-3: New USB device found, idVendor=1546, idProduct=1342, bcdDevice=
0.00
[ 2040.609815] usb 1-3: New USB device strings: Mfr=3, Product=2, SerialNumber=4
[ 2040.609826] usb 1-3: Product: u-blox Modem
[ 2040.609834] usb 1-3: Manufacturer: u-blox
[ 2040.609842] usb 1-3: SerialNumber: 54e3c9a5
[ 2040.621991] option 1-3:1.0: GSM modem (1-port) converter detected
[ 2040.622078] usb 1-3: GSM modem (1-port) converter now attached to ttyUSB0
[ 2040.622141] option 1-3:1.1: GSM modem (1-port) converter detected
[ 2040.622189] usb 1-3: GSM modem (1-port) converter now attached to ttyUSB1
[ 2040.622249] option 1-3:1.2: GSM modem (1-port) converter detected
[ 2040.622324] usb 1-3: GSM modem (1-port) converter now attached to ttyUSB2
[ 2040.622387] option 1-3:1.3: GSM modem (1-port) converter detected
[ 2040.622433] usb 1-3: GSM modem (1-port) converter now attached to ttyUSB3
[ 2040.623595] qmi_wwan 1-3:1.4: cdc-wdm1: USB WDM device
[ 2040.623788] qmi_wwan 1-3:1.4 wwan0: register 'qmi_wwan' at usb-0000:00:14.0-3, WWAN/QMI
device, 2e:df:62:2c:83:ed
…
```

The code snippet above shows that the `option` driver sets up the serial devices `ttyUSB0` to `ttyUSB3`, and `qmi_wwan` is setting up the control interface `cdc-wdm1` and the network interface `wwan0`.

The control interface cdc-wdm is mandatory to set up and to manage the RmNet protocol. Over this USB channel runs the QMI binary protocol.

### 2.1.6 Module control at Userspace level

Linux distributions based on **Ubuntu 22.04** have been used to test and validate RmNet module functionality.

The QMI protocol is used to control the module while in RmNet mode, especially to start and stop a connection and to set up the IP connectivity for the TCP/IP stack.

The command line interface tool, `qmicli`, is used to control the module. In Ubuntu/Debian, it is possible to download pre-built packages:

```
# sudo apt install libqmi-utils
# qmicli –version
qmicli 1.30.4
Copyright (C) 2012-2021 Aleksander Morgado
License GPLv2+: GNU GPL version 2 or later <http://gnu.org/licenses/gpl-2.0.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

The proper version to use is the release 1.30.4 and is available for any major Linux distribution.

☞ There could some existing tools (e.g., `ModemManager`) embedded in major Linux distro. These tools could interfere with DTE-DCE communication, by sending AT commands and using QMI protocol.

☞ Stop and disable at next boot the following common packages, to allow proper modem operations and avoid any interference.

```
# systemctl disable ModemManager.service
# systemctl stop ModemManager.service
# systemctl disable NetworkManager.service
# systemctl stop NetworkManager.service
```

## 2.2 Reference drivers

Find LARA-R6 / LARA-L6 reference drivers for Linux host in the public u-blox GitHub folder [3].

# 3 Using the modem with QMI

QMI is the proprietary protocol for exploiting RmNet on LARA-R6 / LARA-L6 module and to set up the IP connectivity with the mobile network operator (MNO). It can replace the AT commands or be used along with the serial port interface.

There are several QMI commands, also called Messages or APIs. The `qmicli` tool provides a list of API s within sub-system area domains of pertinence. To read official online documentation:

```
# man qmicli
```

Or,

```
# qmicli --help
```

Only a sub-set of APIs are supported LARA-R6 / LARA-L6 series modules. Find reference code in public GitHub repository [3].

Ubuntu 22.04 LTS based distro using Kernels v5.12.x are recommended, as it provides reference **ll6-rmnet** functions/script.

Figure 1 shows how LARA-R6 / LARA-L6 behaves with Linux host in RmNet mode and how TCP/IP traffic, application data, and control are handled.

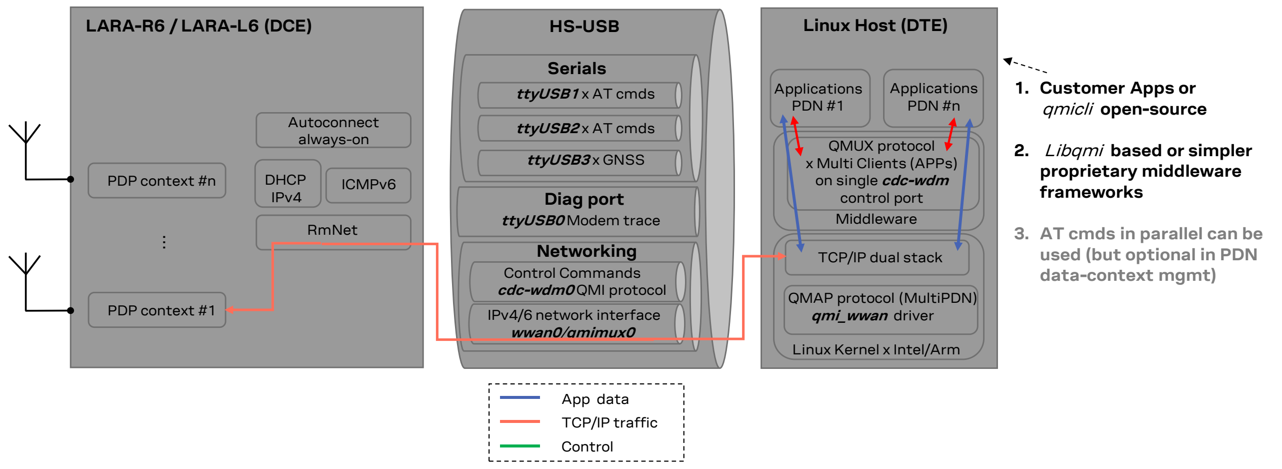


**Figure 1: DCE-DTE block diagram**

☞ RmNet is not the default LARA-R6 / LARA-L6 networking mode. To connect the module in RmNet mode, issue an AT command to switch to this USB networking mode.

AT commands can be sent to the Linux interface using several terminals, e.g., `picocom`. See appendix A for how to set up AT terminal.

Issue the following AT commands to switch to RmNet mode.

```
# picocom /dev/ttyUSB1
ATE0
OK
AT+UUSBCONF=4,"RMNET"
OK
AT+CFUN=16
OK
```

Now the module reboots (changing VID:PID as shown in Table 1),

```
# picocom /dev/ttyUSB1
AT+UUSBCONF?
+UUSBCONF: 4,"RMNET",,"0x1342"
```

```
OK
```

## 3.1 Supported commands

Below is a list of commands grouped by service area:

1. QMI-DMS Device Management Service

- dms-reset
- dms-get-operating-mode
- dms-set-operating-mode=reset
- dms-set-operating-mode=low-power
- dms-set-operating-mode=persistent-low-power
- dms-set-operating-mode=online
- dms-get-ids
- dms-get-capabilities
- dms-get-manufacturer
- dms-get-model
- dms-get-revision

2. QMI-UIM User Identity Module

- uim-get-card-status
- uim-verify-pin
- uim-read-transparent

3. QMI-NAS Network Access Service

- nas-get-home-network
- nas-get-technology-preference
- nas-get-operator-name
- nas-get-system-info
- nas-get-system-selection-preference
- nas-set-system-selection-preference
- nas-get-serving-system
- nas-get-signal-strength
- nas-get-tx-rx-info
- nas-force-network-search
- nas-network-scan

4. QMI-WDA Wireless Data Administrative

- wda-get-data-format
- wda-set-data-format

5. QMI-WDS Wireless Data Service

- wds-noop
- wds-reset
- wds-set-ip-family=4
- wds-start-network
- wds-stop-network
- wds-get-current-settings

- wds-get-current-data-bearer-technology
- wds-get-packet-service-status
- wds-get-packet-statistics
- wda-get-data-format
- wda-set-data-format
- wds-bind-mux-data-port
- wds-get-autoconnect-settings
- wds-set-autoconnect-settings
- wds-create-profile
- wds-modify-profile
- wds-delete-profile
- wds-get-profile-list
- wds-get-default-profile-number
- wds-set-default-profile-number
- wds-get-default-settings
- wds-get-lte-attach-parameters

6. QMI related setting for the DTE (are commodity provided by qmicli to the Host)

- set-expected-data-format
- get-expected-data-format
- link-list
- link-add
- link-delete
- link-delete-all

## 3.2 Establishing QMI connection

All the listed steps shall be performed as Linux sudo user.

The qmcli instance can be opened in the cdc-wdm device reported by the kernel dmesg in the usb-enumeration logs.

### 3.2.1 Module information retrieval

To read module information such as IMEI, manufacturer, model, FW version (in the example below LARA-L6 is used):

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --dms-get-ids
[/dev/cdc-wdm1] Device IDs retrieved:
           ESN: '0'
          IMEI: '353500720001165'
          MEID: 'unknown'
       IMEI SV: '1'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --dms-get-
apabilities
[/dev/cdc-wdm1] Device capabilities retrieved:
       Max TX channel rate: '50000000'
       Max RX channel rate: '100000000'
              Data Service: 'non-simultaneous-cs-ps'
                       SIM: 'supported'
                  Networks: 'gsm, umts, lte'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --dms-get-
manufacturer
[/dev/cdc-wdm1] Device manufacturer retrieved:
       Manufacturer: 'U-BLOX AG'
```

```
# qmicli  --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --dms-get-model
[/dev/cdc-wdm1] Device model retrieved:
       Model: 'LARA-L6004D-00B-00'

# qmicli  --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --dms-get-revision
[/dev/cdc-wdm1] Device revision retrieved:
       Revision: '03.15   1   [Mar 15 2022 12:00:00]'
```

### 3.2.2 SIM information retrieval

To read data needed for device identification management, such as IMSI and USIM ICC numbers:

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --uim-get-card-
status
[/dev/cdc-wdm1] Successfully got card status
Provisioning applications:
       Primary GW:   slot '1', application '1'
       Primary 1X:   session doesn't exist
       Secondary GW: session doesn't exist
       Secondary 1X: session doesn't exist
Slot [1]:
       Card state: 'present'
       UPIN state: 'not-initialized'
              UPIN retries: '0'
              UPUK retries: '0'
       Application [1]:
              Application type:  'usim (2)'
              Application state: 'ready'
              Application ID:
                     A0:00:00:00:87:10:02:FF:FF:FF:FF:89:06:04:00:FF
              Personalization state: 'ready'
              UPIN replaces PIN1: 'no'
              PIN1 state: 'disabled'
                     PIN1 retries: '3'
                     PUK1 retries: '10'
              PIN2 state: 'enabled-not-verified'
                     PIN2 retries: '3'
                     PUK2 retries: '10'
       Application [2]:
              Application type:  'unknown (0)'
              Application state: 'detected'
              Application ID:
                     A0:00:54:45:4C:45:43:4F:4D:49:54:41:B0:01:10:00
              Personalization state: 'unknown'
              UPIN replaces PIN1: 'no'
              PIN1 state: 'not-initialized'
                     PIN1 retries: '0'
                     PUK1 retries: '0'
              PIN2 state: 'not-initialized'
                     PIN2 retries: '0'
                     PUK2 retries: '0'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --uim-read-
transparent=0x3F00,0x7FFF,0x6F07
[/dev/cdc-wdm1] Successfully read information from the UIM:
Card result:
       SW1: '0x90'
       SW2: '0x00'
Read result:
       08:29:22:10:78:40:83:60:34
```

`222018704380643` is the hex unordered IMSI number.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --uim-read-
transparent=0x3F00,0x2FE2
[/dev/cdc-wdm1] Successfully read information from the UIM:
Card result:
        SW1: '0x90'
        SW2: '0x00'
Read result:
        98:93:10:00:00:22:27:96:27:36
```

`89390100002272697263` is the hex unordered ICC number.

#### 3.2.2.1 PIN Unlocking

If SIM card requires the PIN, use the following commands:

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --uim-get-card-
status
[/dev/cdc-wdm1] Successfully got card status
Provisioning applications:
        Primary GW:   slot '1', application '1'
        Primary 1X:   session doesn't exist
        Secondary GW: session doesn't exist
        Secondary 1X: session doesn't exist
Slot [1]:
        Card state: 'present'
        UPIN state: 'not-initialized'
                UPIN retries: '0'
                UPUK retries: '0'
        Application [1]:
                Application type:  'usim (2)'
                Application state: 'pin1-or-upin-pin-required'
                Application ID:
                        A0:00:00:00:87:10:02:FF:39:FF:FF:89:01:01:01:00
                Personalization state: 'unknown'
                UPIN replaces PIN1: 'no'
                PIN1 state: 'enabled-not-verified'
                        PIN1 retries: '3'
                        PUK1 retries: '10'
                PIN2 state: 'enabled-not-verified'
                        PIN2 retries: '3'
                        PUK2 retries: '10'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --uim-verify-
pin=PIN1,1234
[/dev/cdc-wdm1] PIN verified successfully
```

## 3.3 Establishing single PDN connection (IPv4)

To provide a single connection, configure the following setting of the module:

- Raw-IP only mode
- No QoS support
- Host MTU size as provided by the modem
- QMAP protocol disabled

The auto-connection feature can be used and support the same setup of a single PDN connection.

☞ PDN profile #1 is the LTE default bearer and is used for the MNO's auto attachment purposes. Also, other PDN context like `ims` or `sos` can be automatically connected by the modem itself.

Use the `+CGCONT` read command to check the full PDN connection list used by the modem firmware.

Only the default-bearer status can be retrieved via QMI command.

In this scenario Single-PDN means a single user-PDN, e.g., for internet access.

☞ The `wwan0` network interface used below must be the same as reported by kernel `dmesg` usb-enumeration logs.

The code snippet below shows the mandatory setting and final read-out of Linux sysfs value for verification purpose.

```
# ip link set dev wwan0 down
# qmicli --device-open-sync --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1
--wda-set-data-format="link-layer-protocol=raw-ip,ul-protocol=disabled,dl-
protocol=disabled,ep-type=hsusb,ep-iface-number=2"
[/dev/cdc-wdm1] Successfully set data format
                        QoS flow header: no
                    Link layer protocol: 'raw-ip'
       Uplink data aggregation protocol: 'disabled'
     Downlink data aggregation protocol: 'disabled'
                          NDP signature: '0'
Downlink data aggregation max datagrams: '0'
     Downlink data aggregation max size: '0'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --set-expected-data-
format=raw-ip
[/dev/cdc-wdm1] expected data format set to: raw-ip

# cat /sys/class/net/wwan0/qmi/raw_ip
Y
```

## 3.3.1 Setup using explicit APN string

Providing the APN string explicitly in the QMI API is allowed, but not suggested, because mixing modem usage via AT control interface is not reflecting the connection status with the MNO.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid  --wds-noop
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-set-ip-family=4
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-start-network="ip-type=4,apn=ibox.tim.it"
[/dev/cdc-wdm1] Network started
       Packet data handle: '2228609792'
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-get-current-settings
[/dev/cdc-wdm1] Current settings retrieved:
           IP Family: IPv4
        IPv4 address: 10.44.14.56
    IPv4 subnet mask: 255.255.255.240
IPv4 gateway address: 10.44.14.57
    IPv4 primary DNS: 217.200.201.67
  IPv4 secondary DNS: 217.200.201.66
                 MTU: 1500
             Domains: none
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'
```

☞ To get a successful PDN up and running, get the CID (Client ID) allocated by the modem WDS service and then use the same ID on every sub-sequence PDN related QMI WDS service command to be given.

☞ Save the `packet data handle` value, which is mandatory needed later for a successful PDN disconnection command.

## 3.3.2 Setup based on PDN profile

It is preferred to define a list of profiles to be used for PDN activation. In this case the defined and activated profile can be in sync with what reported by `+CGDCONT` and `+CGACT` read commands in case of dual interface (AT and QMI) customer application usage.

The following commands are an example on how to create a PDN profile with index #2 (the next free ones are provided by the modem firmware).

It is suggested to create a few empty profiles in a row, then just edit them later by providing the profile index in accordance with connectivity management at application level.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --wds-create-
profile=3gpp
New profile created:
       Profile type: '3gpp'
       Profile index: '2'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --wds-modify-
profile=3gpp,2,”name=myProfile,apn=internet”

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1  --wds-get-profile-
list=3gpp
Profile list retrieved:
       …
       [2] 3gpp - myProfile
              APN: 'internet'
              PDP type: 'ipv4-or-ipv6'
              PDP context number: '2'
              Username: ''
              Password: ''
              Auth: 'none'
              No roaming: 'no'
              APN disabled: 'no'
```

Now to start the PDN activation:

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-start-network=”3gpp-profile=2”
[/dev/cdc-wdm1] Network started
       Packet data handle: '2228609792'
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'
```

## 3.3.3 Linux IPv4 stack setting

The user shall provide the proper IP routing and DNS setting, based on values reported in the API `--wds-get-current-settings` response.

The following example of commands use standard userspace tool to transfer data over modem connectivity.

```
# ip route flush table main
# ip addr flush dev wwan0
# ip link set wwan0 mtu 1500 up
# ip addr add 10.44.14.56/28 dev wwan0
# ip route add default via 10.44.14.57 table main proto static
# ip route flush cache
# ping 8.8.8.8 -I wwan0 -c1
# resolvectl dns wwan0 217.200.201.67 217.200.201.66
# ping u-blox.com -I wwan0 -c1
```

Both `ping` shall have now received back its replies. If a Linux host PC is used as DTE, the default network interface like Eth/WiFi shall be put in DOWN state to not interfere at DNS nameserver priority level.

### 3.3.4 Modem packet status and statistics

It is possible to retrieve PDN status about RAT in use and data transfer statistics:

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-get-packet-service-status
[/dev/cdc-wdm1] Connection status: 'connected'
[/dev/cdc-wdm1] Client ID not released:
        Service: 'wds'
            CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-get-current-data-bearer-technology
[/dev/cdc-wdm1] Data bearer technology (current):
              Network type: '3gpp'
   Radio Access Technology: 'lte'
[/dev/cdc-wdm1] Client ID not released:
        Service: 'wds'
            CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-get-packet-statistics
[/dev/cdc-wdm1] Connection statistics:
        TX packets OK: 4590
        RX packets OK: 5286
        TX packets dropped: 0
        RX packets dropped: 0
        TX bytes OK: 956761
        RX bytes OK: 2713424
[/dev/cdc-wdm1] Client ID not released:
        Service: 'wds'
            CID: '3'
```

### 3.3.5 PDN disconnection

Provide the `Packet data handle` saved during PDN set-up, as described in section 3.3.1. Avoid QMI long timeout response when `'disconnected'` status has already reported in case of a sent stop request.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-get-packet-service-status
[/dev/cdc-wdm1] Connection status: 'connected'
[/dev/cdc-wdm1] Client ID not released:
        Service: 'wds'
            CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-cid=3 --
wds-stop-network=2228609792
Network cancelled... releasing resources
[/dev/cdc-wdm1] Network stopped
```

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --wds-get-packet-
service-status
[/dev/cdc-wdm1] Connection status: 'disconnected'
```

## 3.4 Establishing multiple PDN connections

To set up maximum eight PDN connections, enable the QMAP protocol embedded in the Linux `qmi_wwan` driver and configure the modem properly. The setup sequence below must be exactly followed:

```
# ip link set dev wwan0 down
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --wds-reset
[/dev/cdc-wdm1] Successfully performed WDS service reset

# qmicli --device-open-sync --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1
--wda-set-data-format=”link-layer-protocol=raw-ip,ul-protocol=qmap,dl-protocol=qmap,dl-
max-datagrams=10,dl-datagram-max-size=4096,ep-type=hsusb,ep-iface-number=4”
[/dev/cdc-wdm1] Successfully set data format
                        QoS flow header: no
                    Link layer protocol: 'raw-ip'
       Uplink data aggregation protocol: 'qmap'
     Downlink data aggregation protocol: 'qmap'
                          NDP signature: '0'
Downlink data aggregation max datagrams: '10'
     Downlink data aggregation max size: '4096'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --set-expected-data-
format=raw-ip
[/dev/cdc-wdm1] expected data format set to: raw-ip

# cat /sys/class/net/wwan0/qmi/raw_ip
Y
# cat /sys/class/net/wwan0/mtu
4096
```

Above the mandatory setting to use and then final read-out of Linux sysfs value, as a double-checking purpose.

### 3.4.1 Data connection Setup using QMAP

A first PDN establishment example by using PDN profile configuration. The `qmimux0` network interface is created as a networking Linux interface and the `mux-id` is then used for the follower binding command.

☞ `mux-id` value must be greater than, or equal to 129 to establish RmNet connection.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --link-
add=iface=wwan0,mux-id=129
[/dev/cdc-wdm1] link successfully added:
  iface name: qmimux0
  mux-id:     129

# qmicli  --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-
release-cid --wds-noop
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-bind-mux-data-port=”mux-id=129,ep-iface-number=2”
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'
```

```
# qmicli  --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-
release-cid --client-cid=3 --wds-set-ip-family=4
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli  --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-
release-cid --client-cid=3 --wds-start-network=3gpp-profile=2
[/dev/cdc-wdm1] Network started
       Packet data handle: '2228586128'
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli  --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-
release-cid --client-cid=3 --wds-get-current-settings
[/dev/cdc-wdm1] Current settings retrieved:
           IP Family: IPv4
        IPv4 address: 100.106.23.133
    IPv4 subnet mask: 255.255.255.252
IPv4 gateway address: 100.106.23.134
    IPv4 primary DNS: 10.133.47.94
  IPv4 secondary DNS: 10.132.100.181
                 MTU: 1500
             Domains: none
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'
```

Now a second PDN establishment example by using a PDN explicit configuration. The `qmimux1` network interface is created as a networking Linux interface and the `mux-id` is then used for the follower binding command.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --link-
add=iface=wwan0,mux-id=130
[/dev/cdc-wdm1] link successfully added:
  iface name: qmimux1
  mux-id:     130

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --wds-noop
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '4'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=4 --wds-bind-mux-data-port="mux-id=130,ep-iface-number=2"
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '4'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=4 --wds-set-ip-family=4
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '4'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=4 --wds-start-network="ip-type=4,apn=internet.apn"
[/dev/cdc-wdm1] Network started
       Packet data handle: '2228559520'
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '4'
```

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm0 --client-no-release-
cid --client-cid=4 --wds-get-current-settings
[/dev/cdc-wdm1] Current settings retrieved:
           IP Family: IPv4
        IPv4 address: 176.247.45.73
    IPv4 subnet mask: 255.255.255.252
IPv4 gateway address: 176.247.45.74
    IPv4 primary DNS: 10.133.47.86
  IPv4 secondary DNS: 10.132.100.212
                 MTU: 1500
             Domains: none
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '4'
```

☞ Always use the proper CID value on any QMI command to get connection status or statistics for addressing the correct PDN.

More PDNs can be established concurrently and `qmimuxN` and `mux-id` will increase.

#### 3.4.1.1 Linux IPv4 stack setting

The following sequence of Linux commands example use standard userspace tool to transfer data over modem connectivity. Two specific routing table are created, each with specific subnet range. They must be managed by the user if automatic and specific PDN related routing is needed. The PDN ad-hoc routing table are not needed for a basic ping usage.

```
# ip route flush table main
# ip addr add flush dev qmimux0
# ip addr add flush dev qmimux1
# ip link set wwan0 up
# ip link show
…
43: wwan0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 4096 qdisc fq_codel state UNKNOWN
mode DEFAULT group default qlen 1000
    link/none
44: qmimux0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/none
45: qmimux1: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/none

# ip link set qmimux0 mtu 1500 up
# ip link set qmimux1 mtu 1500 up

# ip addr add 100.106.23.133/30 dev qmimux0
# ip route
100.106.23.132/30 dev qmimux0 proto kernel scope lihnk src 100.106.23.133

# ip addr add 176.247.145.73/30 dev qmimux1
# ip route
176.247.145.72/30 dev qmimux1 proto kernel scope link src 176.247.145.73

# ip route add 100.106.23.132/30 dev qmimux0 src 100.106.23.133 table route-pdn1-table
# ip route add 176.247.145.72/30 dev qmimux1 src 176.247.145.73 table route-pdn2-table

# ip route add default via 100.106.23.134 dev qmimux0 table route-pdn1-table proto static
# ip route add default via 176.247.145.74 dev qmimux1 table route-pdn2-table proto static

# ip rule add from 100.106.23.132/30 table route-pdn1-table
# ip rule add to 100.106.23.132/30 table route-pdn1-table
# ip rule add from 176.247.145.72/30 table route-pdn2-table
# ip rule add to 176.247.145.72/30 table route-pdn2-table
# ip addr show
…
46: wwan0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 4096 qdisc fq_codel state UNKNOWN
group default qlen 1000
    link/none
47: qmimux0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 100.106.23.133/30 scope global qmimux0
       valid_lft forever preferred_lft forever
48: qmimux1: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 176.247.145.73/30 scope global qmimux1
       valid_lft forever preferred_lft forever

# ip route show table route-pdn1-table
default via 100.106.23.134 dev qmimux0 proto static
100.106.23.132/30 dev qmimux0 scope link src 100.106.23.133

# ip route show table route-pdn2-table
default via 176.247.145.74 dev qmimux1 proto static
176.247.145.72/30 dev qmimux1 scope link src 176.247.145.73

# ip route flush cache
# ping 8.8.8.8 -I qmimux0 -c1
# ping 8.8.4.4 -I qmimux1 -c1
# resolvectl dns qmimux0 10.133.47.94 10.132.100.181
# resolvectl dns qmimux1 10.133.47.86 10.132.100.212
# ping u-blox.com -I qmimux0 -c1
# ping u-blox.com -I qmimux1 -c1
```

Both `ping` shall have now received back its replies. Check specific packets statistics by the proper QMI command on each CID (PDNs).

☞ If a Linux host PC is used as DTE, the default network interface like Eth/WiFi must be put in DOWN state to not interfere at DNS nameserver priority level.

#### 3.4.1.2 QMAP disconnection

Provide the `Packet data handle` saved during PDN setup, as described in section 3.3.1. Avoid QMI long timeout response when `'disconnected'` status has already reported in case of a sent stop request.

```
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=3 --wds-get-packet-service-status
[/dev/cdc-wdm1] Connection status: 'connected'
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
           CID: '3'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 wdm1 --client-no-
release-cid --client-cid=3 --wds-stop-network=2228586128
Network cancelled... releasing resources
[/dev/cdc-wdm1] Network stopped

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-cid=3 --
wds-get-packet-service-status
[/dev/cdc-wdm1] Connection status: 'disconnected'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-no-release-
cid --client-cid=4 --wds-get-packet-service-status
[/dev/cdc-wdm1] Connection status: 'connected'
[/dev/cdc-wdm1] Client ID not released:
       Service: 'wds'
```

```
           CID: '4'

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 wdm1 --client-no-
release-cid --client-cid=4 --wds-stop-network=2228559520
Network cancelled... releasing resources
[/dev/cdc-wdm1] Network stopped

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --client-cid=4 --
wds-get-packet-service-status
[/dev/cdc-wdm1] Connection status: 'disconnected'


Besides for a correct workflow, please remember to clean-up also the Linux IP stack side
with the following steps.
# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --link-list=wwan0
[/dev/cdc-wdm1] found 2 links:
       [0] qmimux0
       [1] qmimux1

# ip addr flush dev qmimux0
# ip addr flush dev qmimux1
# ip route flush table route-pdn1-table
# ip route flush table route-pdn2-table
# ip rule del from all table route-pdn1-table
# ip rule del from all table route-pdn2-table
# ip link set qmimux0 down
# ip link set qmimux1 down

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --link-delete="link-
iface=qmimux0,mux-id=1"
[/dev/cdc-wdm1] link successfully deleted

# qmicli --device-open-proxy --device-open-qmi --device=/dev/cdc-wdm1 --link-delete="link-
iface=qmimux1,mux-id=2"
[/dev/cdc-wdm1] link successfully deleted
```

## 3.5 Reference scripts

Find LARA-R6 / LARA-L6 reference drivers for Linux host in the public u-blox GitHub folder [3]. The reference driver provides a quick and safe experience with QMI and RMNET. The users can easily perform the most normal operation with a cellular modem.

It shall be an out-of-the-box experience for any latest Ubuntu/Debian based distro and are provided as `bash/zsh` scripts to be sourced within a terminal shell.

Once the script is sourced, run the following command to see the content available to the user

```
# lx6help -h
# lx6help
```

Other feature supported by these reference scripts are:

- Resetting/rebooting the module
- Putting the module offline/online
- PDN profile management
- PDN autoconnection - module firmware feature to be used with a DHCP client
- Viewing script usage within the Linux logging system

# 4 Using the modem with ECM

ECM is a protocol used to send and receive Ethernet frames over a USB. It can be used in conjunction with AT command ports. The module used through ECM supports both bridge and router mode.

☞ To use the ECM mode, the module must be properly configurated via AT commands.

With the installed serial terminal (see appendix A) switch to ECM mode using the +UUSBCONF AT command:

```
# picocom /dev/ttyUSB1
ATE1
OK
AT+UUSBCONF=4,"ECM"
OK
AT+CFUN=16
OK
```

Now the module reboots (changing VID:PID as shown in Table 1),

```
# picocom /dev/ttyUSB1
AT+UUSBCONF?
+UUSBCONF: 4,"ECM",,"0x1343"
OK
```

## 4.1 Basic implementation

The Connection Manager (CM) configures the DTE by considering the status of each PDP context/EPS bearer, retrieved with +CGDCONT and +CGACT AT commands.

The CM must properly set the IP, gateway and DNS addresses for the CDC-ECM interface aliases/virtual interfaces. Once the PDP context/EPS bearers are activated and the configuration is set, the CM must periodically poll the target to update the settings or to track state changes of the PDP contexts, caused, e.g., by the deactivation of PDP contexts commanded by the network due to roaming or temporary network issues.

The following list of instructions can be used as reference for the implementation on IPv4 based networks:

- In 2G/3G: define a PDP context with +CGDCONT AT command. In 4G: the default initial PDP context is automatically activated during attach.
- Verify if the module is attached to the network by the +CREG, +CGREG and +CEREG URCs.
- Set up the OS with IP alias/virtual interfaces, routing rules and DNS configuration.
- If the PDP context is deactivated (it can be inferred by the information URCs using the +CGEREP AT command) then remove all the settings for the related interface alias/virtual interface.

## 4.2 Enable data connection to ECM

In ECM mode the ECM data connection must be enabled in two manners:

- **Manually** at every boot or network change.
- **Automatically** with an auto connect configuration.

### 4.2.1 Enable data connection to ECM manually

| Command | Response | Description |
|---|---|---|
| AT+UIFCONF=2,0,0 | +UIFCONF: 0<br>OK | Retrieve the ECM data connection status |
| AT+UIFCONF=2,0,0,1 | OK | Enable the ECM data connection |

### 4.2.2 Enable data connection to ECM automatically

ECM data connection can be enabled automatically using the auto connect configuration.

| Command | Response | Description |
|---|---|---|
| `AT+UIFCONF=2,0,11` | `+UIFCONF: 0`<br>`OK` | Retrieve the automatic ECM data connection status |
| `AT+UIFCONF=2,0,11,1` | `OK` | Enable the automatic ECM data connection |
| `AT+UIFCONF=2,0,100` | `OK` | Save the autoconnection configuration |
| `AT+CFUN=16` | `OK` | Reboot the module |

☞ By default, interface roaming configuration is disabled. To enable it, refer to the `+UIFCONF` AT command in the LARA-L6 / LARA-R6 series AT commands manual [1].

## 4.3 Bridge mode

In bridge mode, the target acts as a bridge device between the mobile network and the DTE: the IP termination of the data connection is on the DTE network subsystem. For the activated PDP context or EPS bearer, the DTE assigns (i.e., “binds”) the IP address to its USB virtual Ethernet interface and configures its routing rules. Each IP address associated with an active PDP context/EPS bearer is granted to the target by the mobile network, and it shall be retrieved through appropriate AT commands. In bridge mode the PDP context/EPS bearer sets up a bridge between the cellular network and the USB interface: this is defined as bridge PDP context/EPS bearer.

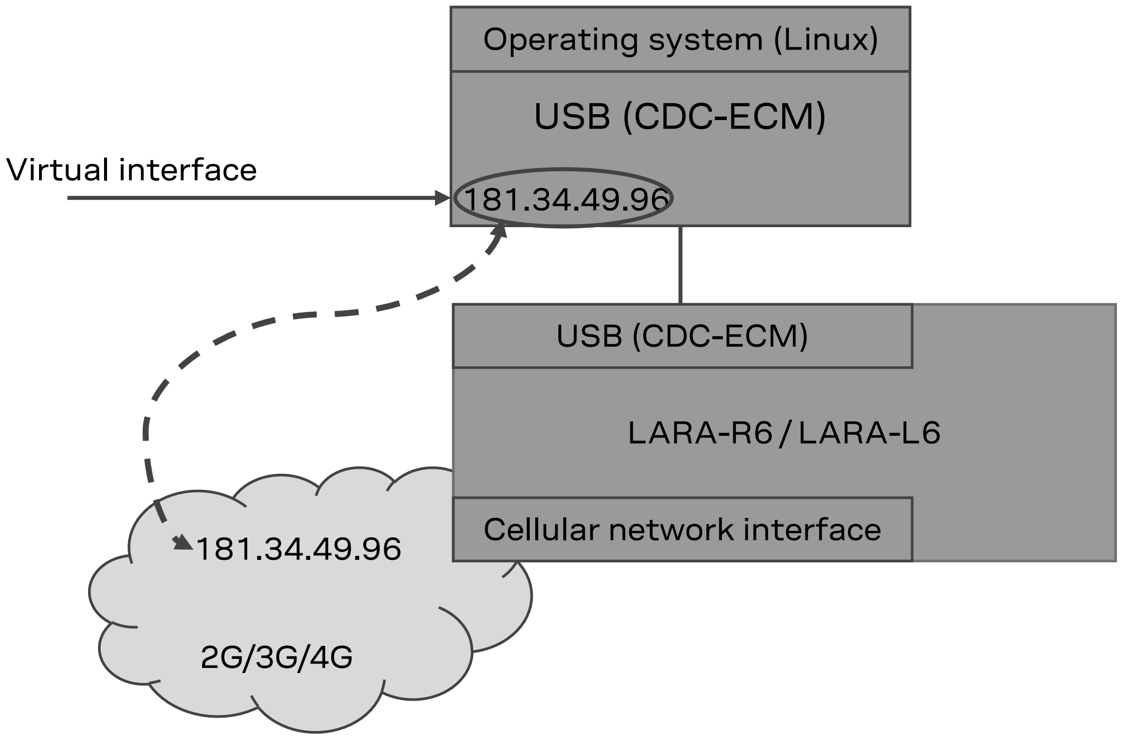


**Figure 2: LARA-R6 / LARA-L6 ECM bridge mode**

### 4.3.1 Set the module to bridge mode

| Command | Response | Description |
|---|---|---|
| `AT+UIFCONF=2,0,10` | `+UIFCONF: 1`<br>`OK` | Retrieve the ECM module router/bridge mode configuration |

| Command | Response | Description |
|---|---|---|
| `AT+UIFCONF=2,0,10,0` | `OK` | Set the module to bridge mode configuration |
| `AT+UIFCONF=2,0,100` | `OK` | Save the bridge mode configuration |
| `AT+CFUN=16` | `OK` | Reboot the module |

## 4.4 Router mode

In the router mode, the IP termination is on the target: the module acts as a mobile network router and it can share its data connectivity through a private network over the USB interface. In router mode, the PDP context/EPS bearer are connected to the module IP subsystem: this is defined as router PDP context/EPS bearer.

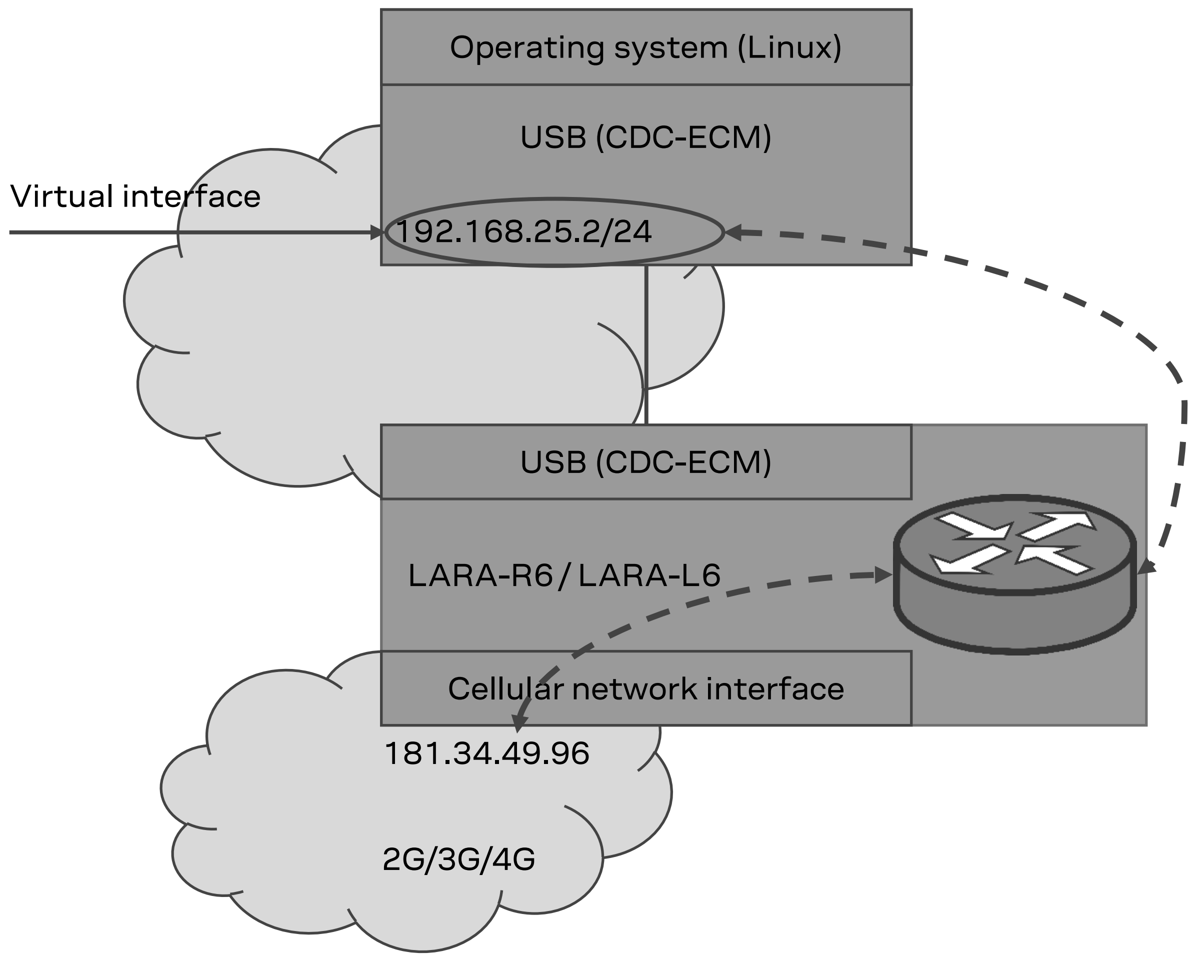


**Figure 3: LARA-R6 / LARA-L6 ECM router mode**

### 4.4.1 Set the module to router mode

☞ Router mode configuration factory-programmed value is set to router mode.

| Command | Response | Description |
|---|---|---|
| `AT+UIFCONF=2,0,10` | `+UIFCONF: 0`<br>`OK` | Retrieve the ECM module router/bridge mode configuration |
| `AT+UIFCONF=2,0,10,1` | `OK` | Set the module to router mode configuration |
| `AT+UIFCONF=2,0,100` | `OK` | Save the bridge mode configuration |
| `AT+CFUN=16` | `OK` | Reboot the module |

## 4.5 IPv4 stack setting

The following Linux methods use standard userspace tools to transfer data over modem connectivity.

### 4.5.1 IPv4 stack setting manually using ip command

The example below uses `ip` command to set the network parameters.

```
# ip route flush table main
# ip addr flush dev wwan0
# ip link set wwan0 mtu <MTU> up
# ip addr add <ip_addr>/<netmask> dev wwan0
# ip route add default via <default_gateway> table main proto static
# ip route flush cache
# ping 8.8.8.8 -I wwan0 -c1
# resolvectl dns wwan0 <DNS_prim_addr> <DNS_sec_addr>
# ping u-blox.com -I wwan0 -c1
```

#### 4.5.1.1 Bridge mode

The network parameters can be obtained using the AT commands in the following table.

| Parameters | AT command |
|---|---|
| <MTU>, <DNS_prim_addr>, <DNS_sec_addr>, <ip_addr> | `+CGCONTRDP` |
| <default_gateway>, <netmask>, <DNS_prim_addr>, <DNS_sec_addr> | `+UIPADDR` |

☞ The `+UIPADDR` AT command returns the `<default_gateway>` address by `<ipv4_address>` parameter, as IPV4 address is the gateway from the host CPU perspective.

#### 4.5.1.2 Router mode

The network parameters can be obtained using the AT commands in the following table.

| Parameters | AT command |
|---|---|
| <MTU>, <DNS_prim_addr>, <DNS_sec_addr> | `+CGCONTRDP` |
| <default_gateway> | `+UIFCONF=2,0,0,1` |
| <netmask> | `+UIFCONF=2,0,1,1` |

`<ip_addr>` can be obtaining combining `<netmask>` and `<default_gateway>`.

### 4.5.2 IPv4 stack setting using DHCP

Below is an example of the using `udhcpc` command to obtain automatically all the network parameters.

```
# ip link set wwan0 up
# udhcpc -q -f -i wwan0
```

## 4.6 IPv6 stack setting

☞ IPv6 is available only in bridge mode.

☞ Linux interface is automatically configured and IPv6 address is automatically obtained in Linux.

To check the IPv6 address, use the command below with the correct obtained interface:

```
# ip -6 addr show dev <interface>
```

The following example shows the output of the `# ip -6 addr show dev wwan0` command:

```
9: wwan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group
default qlen 1000
```

```
       inet6 2001::2:3711:ec08:7206:84f/64 scope global temporary dynamic
              valid_lft 604770sec preferred_lft 86273sec
       inet6 2001::2:a695:5940:7429:cf1e/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 2591969sec preferred_lft 604769sec
       inet6 fe80::6113:ae3a:3076:947f/64 scope link
       valid_lft forever preferred_lft forever
```

The command output example above shows that there are several automatically obtained `inet6`, indicating that the IPv6 is correctly configured.

# Appendix

# A Setting up AT terminal for LARA-R6 / LARA-L6 series modules

AT commands can be sent to the Linux interface using several terminals. In this document we refer to `picocom`. Download `picocom` serial terminal to ease the AT command mode input and output. For the complete list of available AT commands, see the LARA-L6 / LARA-R6 series AT commands manual [1].

```
# apt install picocom
$ sudo usermod -a -G dialog $USER
```

# B Glossary

| Abbreviation | Definition |
|---|---|
| BER | Bit Error Rate |
| CDC | Communications Device Class |
| CM | Connection manager |
| DCE | Data Communication Equipment |
| DTE | Data Terminal Equipment |
| DRX | Discontinuous Reception |
| DDC | Display Data Channel |
| ECM | Ethernet Control Model |
| MTU | Maximum Transmission Unit |
| PDN | Public Data Network |
| PDP | Packet Data Protocol |
| RA | Router Advertisement |

# Related documentation

[1] u-blox LARA-L6 / LARA-R6 series AT commands manual, UBX-21046719

[2] u-blox LARA-L6 / LARA-R6 series application development guide, UBX-22001850

[3] u-blox public GitHub page, https://github.com/u-blox

☞ For product change notifications and regular updates of u-blox documentation, register on our website, www.u-blox.com.

# Revision history

| Revision | Date | Name | Comments |
|---|---|---|---|
| R01 | 13-Dec-2022 | dtro/pgel | Initial release |
| R02 | 04-Apr-2023 | dtro/pgel | Updated ECM IPv4 stack setting section. Added QMAP mux-id value. |
| R03 | 29-Nov-2023 | dtro | Added support to LARA-R6 "01B" product version. |

# Contact

**u-blox AG**

| | |
|---|---|
| Address: | Zürcherstrasse 68<br>8800 Thalwil<br>Switzerland |

For further support and contact information, visit us at www.u-blox.com/support.