MediaTek LinkIt™ ONE Developer's Guide

Version: 1.2

Release date: 13 February 2015

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	8 September 2014	Initial release
1.1	4 December 2014	Updated for: • Arduino version support • Clarification of mobile network support (2G)
1.2	13 February 2015	Updated for: • Bluetooth GATT profile support and related classes • SDK version 1.1 release

Table of contents

Lists of tables and figures

1 Introduction

1.1 What is LinkIt?

MediaTek LinkIt™ is a collection of development platforms designed for the creation and prototyping of Wearables and Internet of Things (IoT) devices. These development platforms are offered in two distinct families:

- LinkIt ONE, for simple application use wearables and IoT devices such as smart wristband, smart safety and tracking devices. These devices provide the user with feedback and control options on the device, and can exchange data and control messages with users, other smart devices, and cloud applications using GSM messaging, GPRS, Wi-Fi or Bluetooth connections.
- LinkIt Connect, for one application use IoT devices such as smart bulbs and smart appliances that are controlled from cloud services or smartphones over Wi-Fi or Bluetooth connections.

Each development platform in turn will offer one or more chipset and API variants designed to meet specific development and device requirements. And to enable the creation and prototyping of devices, each variant includes the below items:

- One or more HDKs to enable the prototyping of devices.
- An SDK to enable the creation of firmware or software for devices.
- One or more hardware reference designs that can be used as the basis for board layouts for the final product.
- Comprehensive documentation, such as API references, developer guides, chipset descriptions and pin-out diagrams.
- Support forums.

1.2 MediaTek LinkIt ONE development platform

The MediaTek LinkIt ONE development platform (see Figure 1) offers a robust yet flexible development platform for wearable and IoT devices. The platform consists of the following:

- System-on-Chip (SoC) MediaTek MT2502 (Aster), the world's smallest commercial SoC for Wearables and IoT devices, and its energy efficient Wi-Fi and GPS companion chipsets.
- LinkIt ONE APIs.
- LinkIt ONE Hardware Development Kit (HDK).
- LinkIt ONE Software Development Kit (SDK).

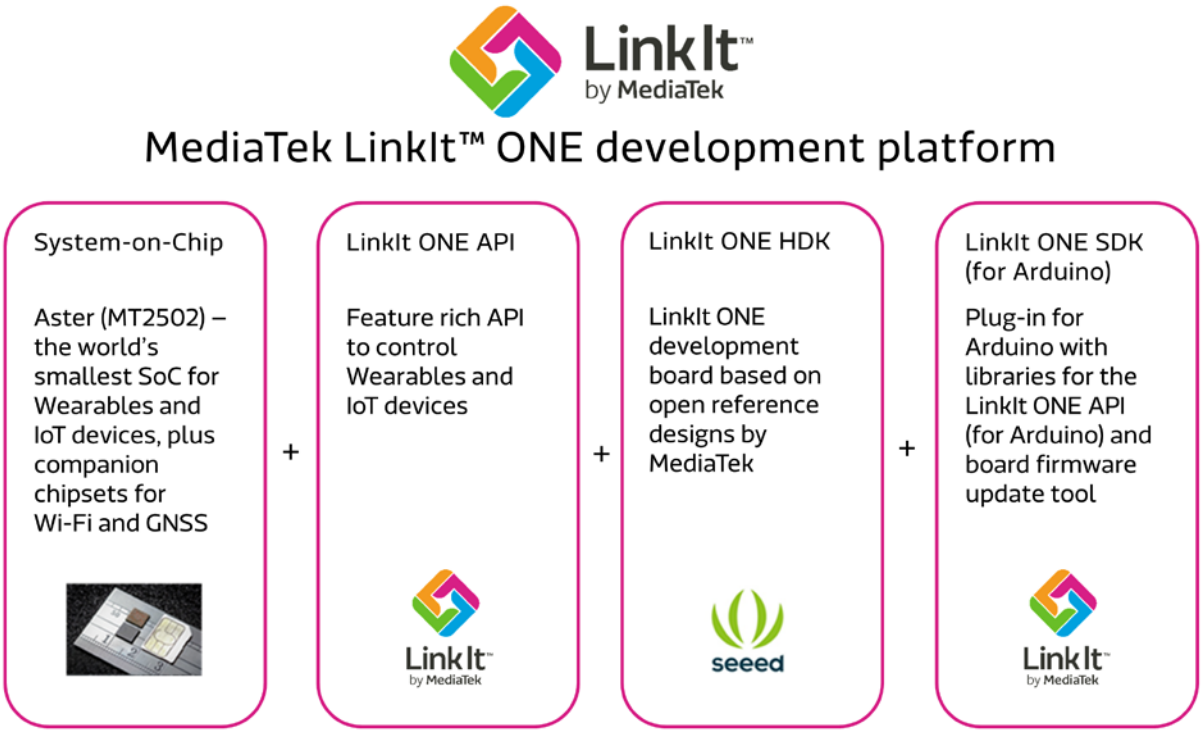

Figure 1 The components of the LinkIt ONE development platform

The LinkIt ONE HDK comprises the LinkIt ONE development board, which provides similar pin-out to Arduino UNO. One of the most feature rich development boards in the market, it's a co-design product of MediaTek and Seeed Studio, the well-known innovation platform. The board provides all the features available to the LinkIt ONE APIs including GPS and Wi-Fi on a single board and offers various interfaces for connecting to sensors and other peripherals.

The LinkIt ONE SDK is a plug-in to the Arduino IDE, the tool of choice in the Maker community. With the SDK you can easily migrate existing Arduino code using the LinkIt ONE APIs. In addition, you get a range of APIs to make use of the LinkIt ONE communication features: 2G mobile network (GSM and GPRS), Bluetooth (2.1 and 4.0) and Wi-Fi.

With the LinkIt ONE SDK and LinkIt ONE development board you've everything you need to prototype innovative products.

If you're not familiar with Arduino, please visit Arduino.cc or the Arduino Playground wiki for more information.

1.3 LinkIt ONE SDK

LinkIt ONE SDK is released as a plug-in for Arduino IDE. The APIs in the SDK provide access to all the connectivity functions provided by LinkIt ONE development platform in addition to core Arduino functions, such as the ability to control digital pins and parse analog sensor inputs. This enables you to build prototypes and demonstrations of connected wearable and IoT devices quickly and easily.

As shown in Figure 2, using the LinkIt ONE SDK you create an Arduino Sketch to make use of the LinkIt ONE APIs. These APIs execute over the run-time environment to enable you to access the features of the LinkIt ONE development board.

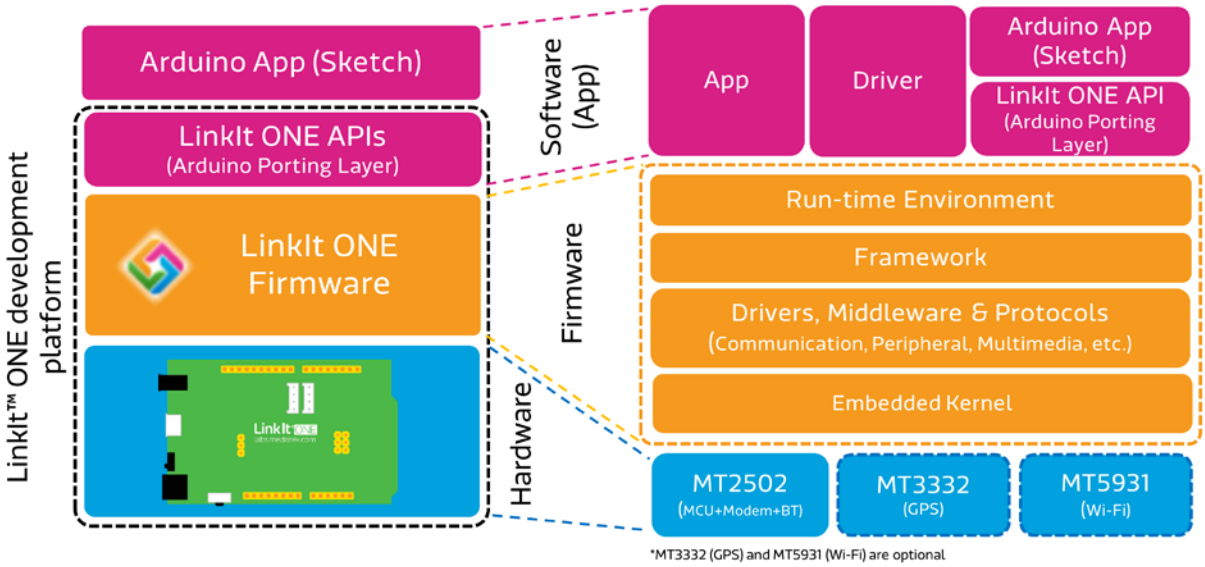

Figure 2 The architecture of the LinkIt ONE development platform

If you're an experienced Arduino developer, you can dive into the MediaTek LinkIt ONE API reference, since most modules conform to the Arduino API style and will be familiar to you.

1.3.1 Sketching your ideas

Developing with LinkIt ONE SDK is as simple as writing an Arduino sketch. A *sketch* is a source code file representing the core controlling logic for the LinkIt ONE development board. It consists of the following:

- A `setup()` function that initializes resources, such as the Wi-Fi module.
- A `loop()` function that continuously listens to and processes events from hardware sensors and software modules such as those for Bluetooth. The `loop()` function runs forever — until the device is shutdown.

1.3.2 Running Arduino sketches

To execute a sketch, LinkIt ONE SDK compiles the sketch into a LinkIt ONE executable (VXP file) as shown in Figure 3. The IDE plug-in then loads the VXP file into the file system of LinkIt ONE development board. When the LinkIt ONE development board boots up, it automatically executes the loaded VXP file. The VXP executable is then loaded by the run-time environment.

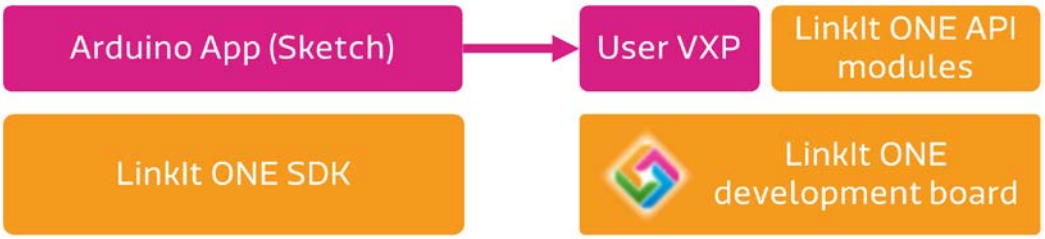

Figure 3 The development and test process

While the sketch itself is written as a single-threaded loop, the run-time environment is a multi-threaded soft-realtime environment. The VXP runs as a standalone thread. The Arduino plug-in wrapper layer is responsible for sending requests to other service modules, which run on separate threads.

1.3.3 LinkIt ONE APIs

LinkIt ONE SDK includes all the LinkIt ONE APIs that can be used with the LinkIt ONE development board. A list of the APIs is given in Table 1.

Arduino Functions	Digital I/O	For the transmission and receipt of digital signals over individual pins.
	Advanced I/O	For the transmission or receipt digital data as pulses or by byte over individual pins.
	Analog I/O	For the transmission or receipt of analog signals through an ADC over individual pins.
	Serial	For the exchange of data with another device over a serial connection.
	Time	For obtaining current time and setting delays.
	Interrupts	For processing external interrupts through an Interrupts Service Routine (ISR).
	Math	Various basic, bits and bytes, random numbers and trigonometry maths functions.
	Servo	For the control of servo motors.
	SPI	For communication with peripherals using the Serial Peripheral Interface (SPI) protocol.
	Wire (I²C)	For communication with peripheral using the Inter-Integrated Circuit (I2C) protocol.
	Stepper	For the control of stepper motors.
LinkIt ONE Connectivity, Storage and Multimedia	GSM	For the composition and sending, and receipt and reading of Short Message Service (SMS) messages.
	GPRS	For connecting to and transferring data over a GPRS (2G) network.
	Storage	For manipulate the file system on an SD card and in the internal Flash memory.
	GPS	For acquiring location information from GPS hardware.
	WiFi	For connecting to and transferring data over a Wi-Fi connection.
	Bluetooth	For connection with and transferring data to and from a Bluetooth enabled device, using the SPP or GATT Bluetooth profiles.
	Audio	To play MP3, AAC and AMR files.
	Battery	To get the battery level and charging state.
	DateTime	To set and get the current date and time information.
	EEPROM	For writing to and reading from the EEPROM provided in LinkIt, enabling data to be stored while the device is powered off.

Table 1 A list of modules available in the LinkIt ONE SDK

Further details on the APIs can be found in Chapter 4, "LinkIt ONE API Guide" and full documentation of the APIs is provided in the MediaTek LinkIt ONE API reference on the MediaTek Labs website.

1.3.4 Extending Your Sketch

Sketch files are written in C++, while the underlying interface to the run-time is in C and the compilation tool chain is GNU gcc. This opens up the possibility of including any open source library written in C and C++ as part of the LinkIt ONE application. However, the SDK only supports a single-threaded programming model — to provide an easy to use environment for hardware prototyping — so it's difficult to port libraries that depend on multi-thread behavior. Another consideration when including external libraries is the RAM size used by your sketch, which should be limited to no more than 2MB of the 4MB RAM on the LinkIt ONE development board.

1.4 Hardware

LinkIt ONE development board is an open source, high performance board. It's based on the world's leading System-0n-Chip (SoC) for wearables MediaTek Aster (MT2502), which is combined with high performance Wi-Fi (MT5931) and GPS (MT3332) chipsets to provide a feature rich development board. It also provides similar pin-out to Arduino UNO for connections to various sensors and peripherals.

The LinkIt ONE development board is a co-design product by Seeed Studio and MediaTek. The design combines the parties' knowledge in open hardware and an industry leading reference design.

1.4.1 LinkIt ONE Pin-out Diagram

LinkIt ONE development board provides a pin-out configuration similar to the Arduino UNO, as shown in Figure 4.

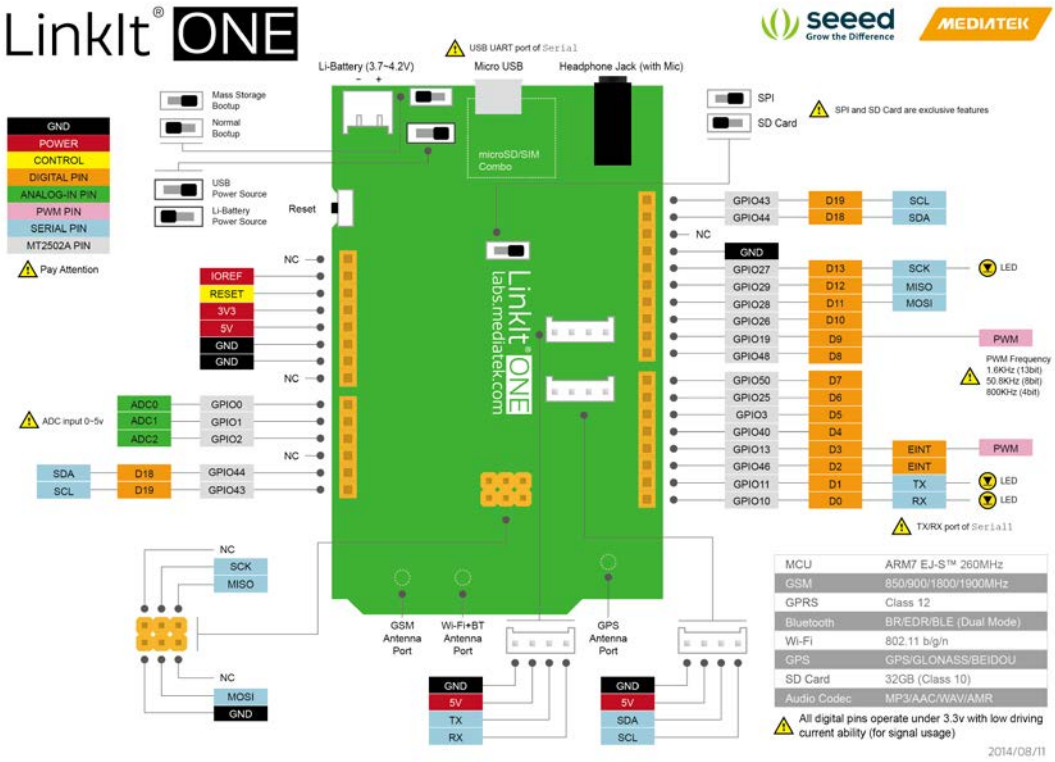

Figure 4 The pin-out configuration of LinkIt ONE development board

1.4.2 LinkIt ONE development board specifications

Table 2 provides specification details of the LinkIt ONE development board.

Table 2 LinkIt ONE development board specifications

Category	Feature	Spec
Microcontroller	Chipset	MT2502A (Aster)
	Core	ARM7 EJ-S™
	Clock Speed	260MHz
PCB Size	Dimensions	3.3 x 2.1 inches
Memory	Flash	16MB
	RAM	4MB
Power	Battery Jack	3.7~4.2V Li-battery
	DC Current Per I/O Pin	0.3~3mA
Digital IO Pins	Pin Count	16 D0~D13, SDA, SCL
	Voltage	3.3v
Analog Input Pins	Pin Count	3 A0, A1, A2
	Voltage	0~5V
PWM Output Pins	Pin Count	2 D3 and D9
	Voltage	3.3v
	Max Resolution	13bit (customizable)
	Max Frequency@Resolution	1.6kHz@13bit 50.8kHz@8bit 800kHz@4bit (customizable)
External Interrupts	Pin Count	2 D2 and D3
I2C (master only)	Set Count	1 SDA, SCL
	Speed	100Kbps, 400Kbps, 3.4Mbps
SPI (master only)	Set Count	1 D11(MOSI), D12(MISO), D13(SCK)
	Speed	104Kbps~26Mbps
UART (Serial1)	Set Count	1 D0(RX), D1(TX)
	Voltage	3.3v
UART on USB (Serial)	Set Count	1
Communications	GSM/GPRS	850/900/1800/1900 MHz
	GPRS	Class 12
	Bluetooth	BR/EDR/BLE (Dual Mode)
	Wi-Fi	802.11 b/g/n
Positioning	GPS	GPS/GLONASS/ BEIDOU
User Storage	Flash	10MB

Category	Feature	Spec
	SD Card	Up to 32GB (Class 10)
Executable Size (Compiled Sketch file)	RAM (Code+RO+RW+ZI+Heap)	2MB

1.5 Joining Our Ecosystem

Wearables and Internet of Things are the next wave in the consumer gadget revolution. MediaTek is a key player in this field, combining the best of two worlds —the existing MediaTek ecosystem of phone manufacturers, electronic device manufacturers and telecom operators combined with the open, vibrant Arduino maker community world. No matter whether you're a maker, device manufacturer, student, DIY hobbyist, or programmer, you can use this powerful yet simple platform to create something innovative. You can join the MediaTek LinkIt ecosystem by registering on labs.mediatek.com, we look forward to you join our ecosystem and creating something great together.

2 Getting started

This section provides a guide to getting started with the LinkIt ONE development platform and covers the following items:

- The supported environments for development.
- Installing the Arduino IDE.
- Installing and configuring the LinkIt ONE SDK.
- Creating the first project.

2.1 Environment

The LinkIt ONE SDK can be used on any edition of Microsoft Windows XP, Vista, 7 and 8.

2.2 Installing Arduino IDE

LinkIt ONE SDK is released as a plug-in for 1.5.6-r2 BETA and 1.5.7 BETA. If you've already installed one of the supported Arduino versions, you can skip this step. If you don't have Arduino installed:

1) Download the Arduino software from the Arduino website.
2) Install the Arduino software.
3) After installation, note the folder in which you installed Arduino, which is the folder containing `arduino.exe` (as shown in Figure 5). You'll install the LinkIt ONE SDK into this folder.

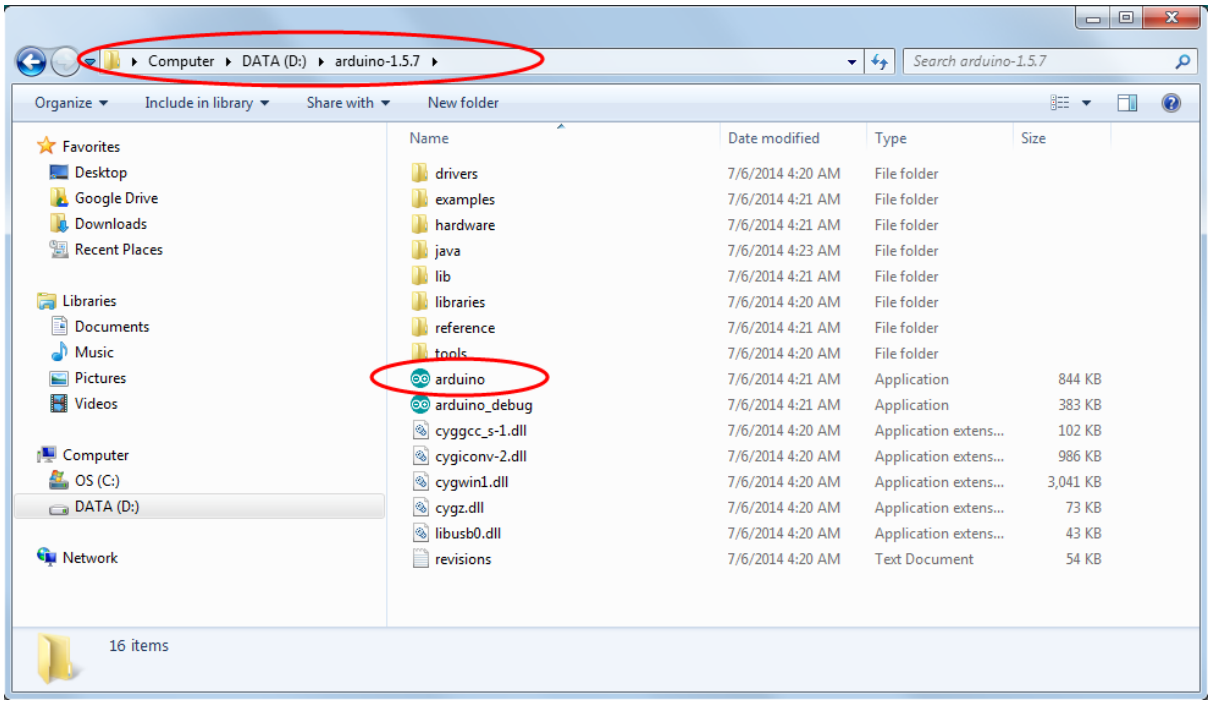

Figure 5 The Arduino installation folder containing Arduino.exe

2.3 Installing LinkIt ONE SDK

To install LinkIt ONE SDK:

1) Register on the MediaTek Labs website.

2) Download the LinkIt ONE SDK zip file from here.
3) Extract the content of the LinkIt ONE SDK zip file.
4) Make sure Arduino IDE is not running.
5) Run the LinkIt ONE SDK installer.
6) In **Select Destination Location**, click **Browse** and locate the folder in which you installed Arduino IDE (the folder you noted earlier) and click **Next**.
7) The SDK installs and on completion the dialog shown in Figure 6 displays. The first time you install the SDK, ensure that **Install the MediaTek USB Driver** is selected, before clicking **Finish**.

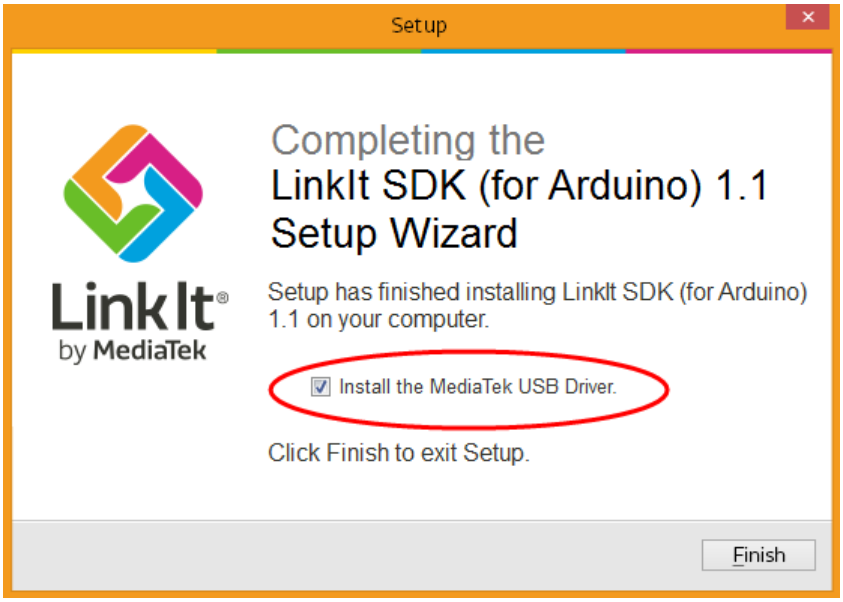

Figure 6 The option to install the MediaTek USB driver should be selected before clicking Finish

The MediaTek USB Driver shouldn't need to be installed again should you upgrade the SDK in the future.

2.4 Configure Arduino IDE to use a LinkIt ONE development board

Now you have LinkIt ONE SDK installed you need to configure Arduino IDE to use your LinkIt ONE development board as follows:

1) Launch Arduino IDE.
2) In Arduino IDE, on the **Tools** menu point to **Board** and click **LinkIt ONE**, as shown in Figure 7, to configure the board as the target.

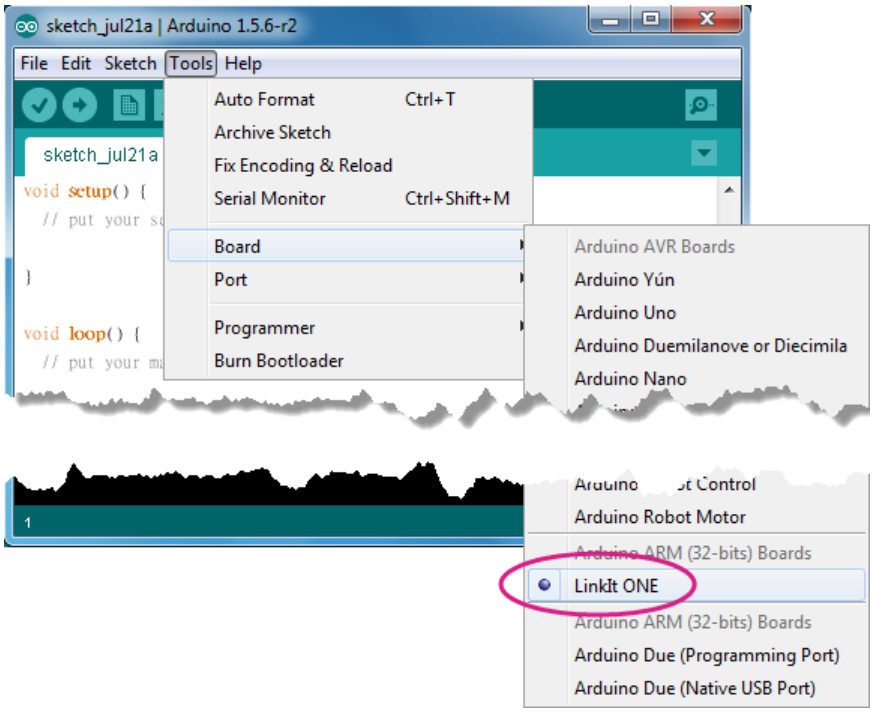

Figure 7 The menu option for configuring the LinkIt ONE development board

3) Open Windows Control Panel then click **System** and:
   - On Windows 7 and 8, click **Device Manager**.

     On Windows XP, click the **Hardware** tab and then **Device Manager**.
4) In Device Manager, navigate to **Ports (COM & LPT)** (see Figure 9).

5) Connect LinkIt ONE development board to your computer using a Micro-USB cable, as shown in Figure 8.

Figure 8 Micro USB connector on the LinkIt ONE development board

6) A new COM device should appear under **Ports (COM & LPT)** in Device Manager, as shown in Figure 9. Note the *COMx* port number of the **MTK USB Debug Port**, this information is needed to complete configuration of Arduino IDE.

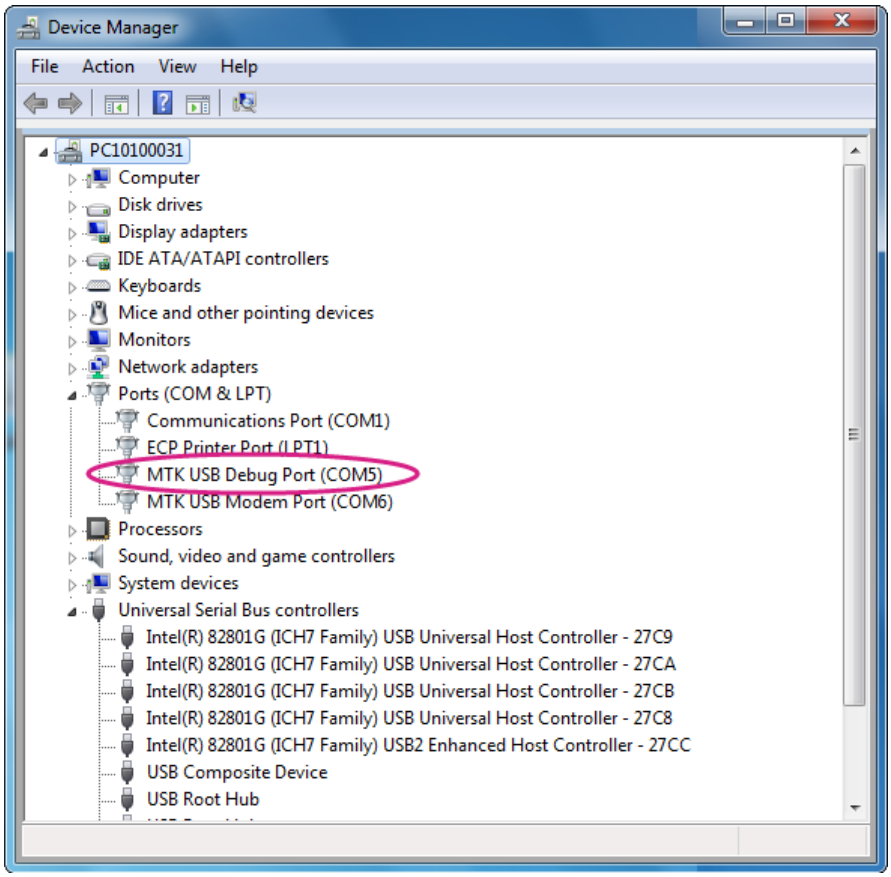

Figure 9 LinkIt ONE development board listed in Device Manager

7) In Arduino IDE on the **Tools** menu, point to **Port**. Click the COMx item that matches the LinkIt ONE development board's port, as shown in Figure 10.

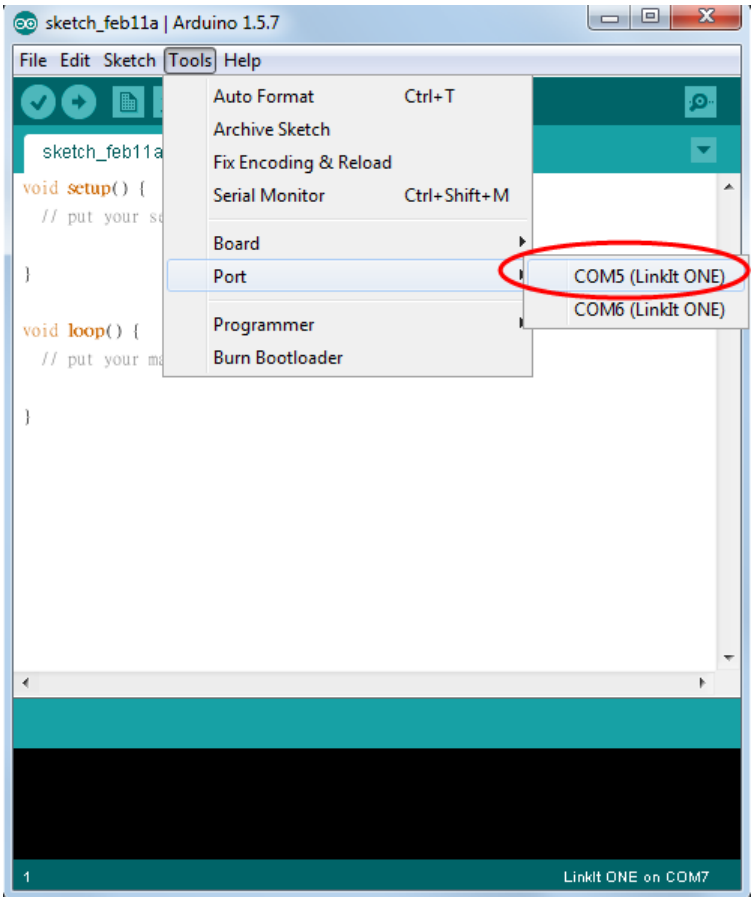

Figure 10 Setting the LinkIt ONE development board's port number in Arduino IDE

Your Arduino IDE is now set up to use the LinkIt ONE SDK and LinkIt ONE development board, you can start creating your first project.

2.5 Your First Project

To create your first LinkIt ONE project:

1) In Arduino IDE on the **File** menu point to **Examples**, then **01. Basics** and click **Blink**, this opens the Blink example.
2) Confirm that your board is in SPI mode, by checking the switch show in Figure 12. The switch should be in the position closest the LED. (For more information, see 4.9, "SPI".)

3) On the **File** menu, click **Upload**, or click the upload icon as shown in Figure 11.

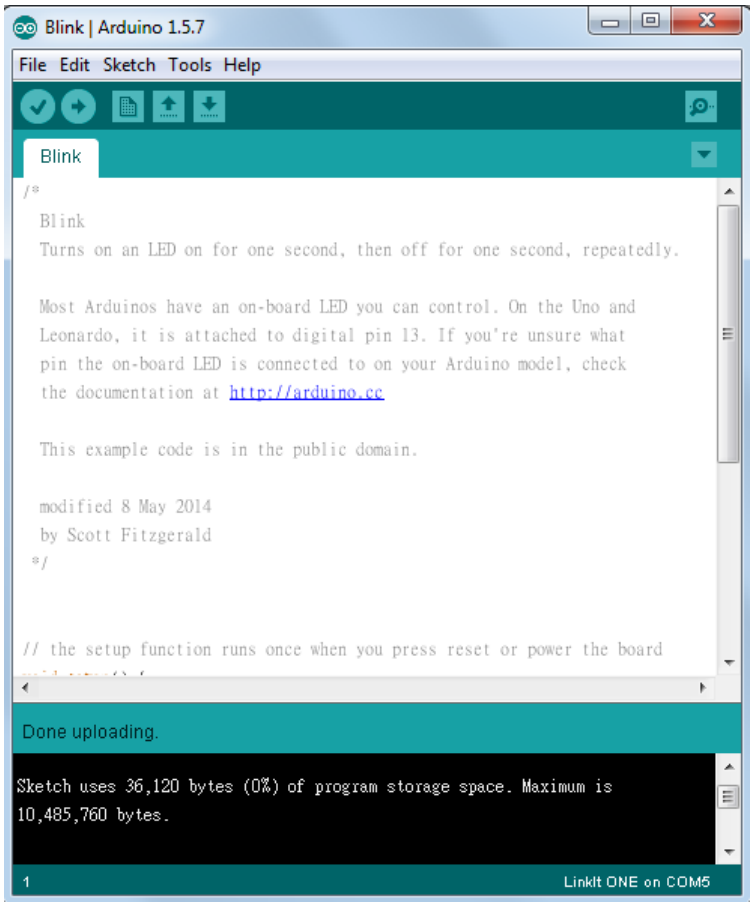

Figure 11 The upload button in toolbar of Arduino IDE

4) Wait for a moment. If everything is correct, an **Upload Done** message will display in Arduino IDE and the LED on LinkIt ONE development board will start to blink on and off every second, as shown in Figure 12.

Figure 12 The LED on the LinkIt ONE board

Congratulations, you've installed LinkIt ONE SDK and run your first LinkIt ONE project, you're now ready to start creating the next generation of Wearable and IoT devices with your LinkIt ONE development board and the LinkIt ONE SDK.

3 Troubleshooting

3.1 Firmware update request

If Arduino IDE reports an error message such as "please upgrade your firmware" when uploading your sketch, take the following steps to update the firmware of LinkIt ONE:

1) If the board isn't already, switch it into mass storage mode, as shown in Figure 13.

Figure 13 LinkIt ONE in mass storage mode

2) Launch LinkIt ONE Firmware Updater, which is located in the Arduino IDE folder under `hardware/tools/mtk`, as shown in Figure 14.

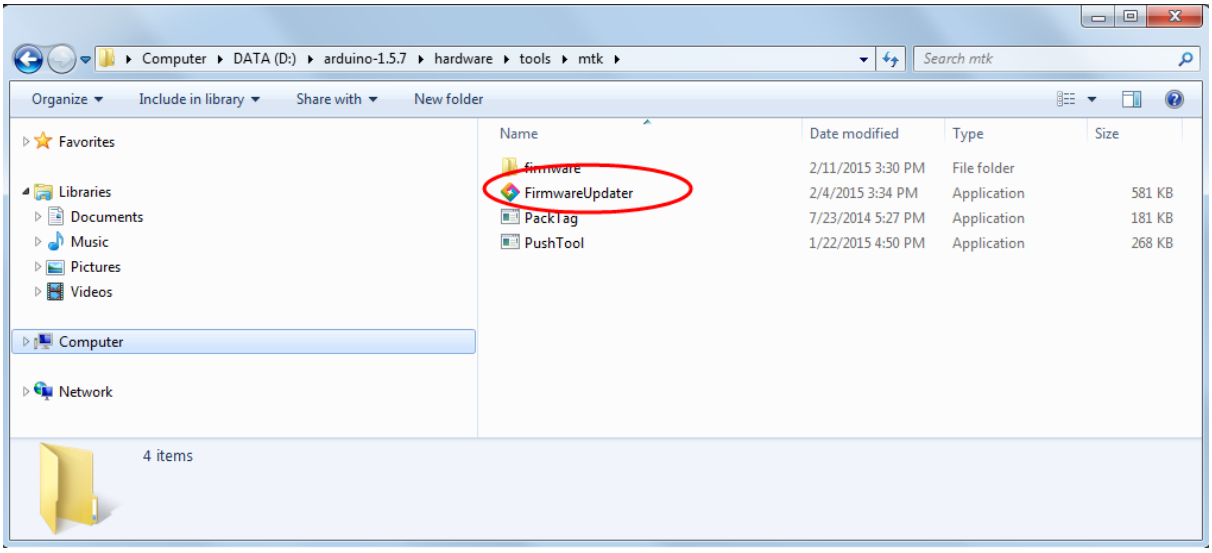

Figure 14 The location of LinkIt ONE Firmware Updater in the Arduino IDE folder

3) In LinkIt ONE Firmware Updater, click the download button, as shown in Figure 15.

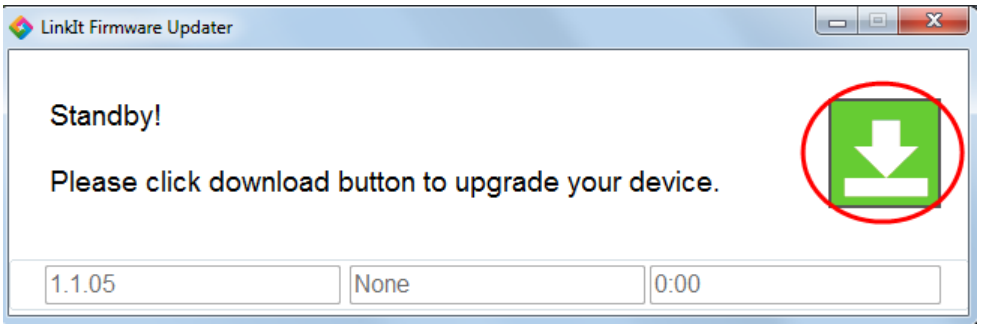

Figure 15 Click the download button to start the firmware update process

4) Follow the instructions, see Figure 16, and disconnect LinkIt ONE from your computer and its power source. Then reconnect the power source and connect LinkIt ONE to your computer using a USB cable.

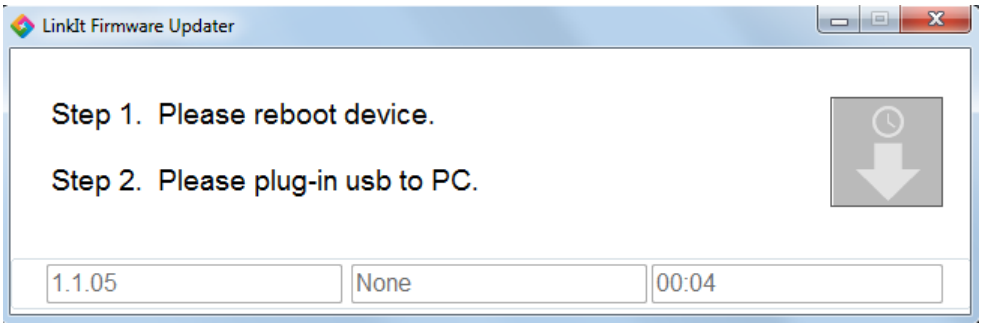

Figure 16 Instructions for upgrading a board's firmware are provided

5) Once LinkIt ONE is reconnected the firmware update will start automatically. LinkIt ONE Firmware Updater will report that the firmware is being downloaded, as shown in Figure 17.

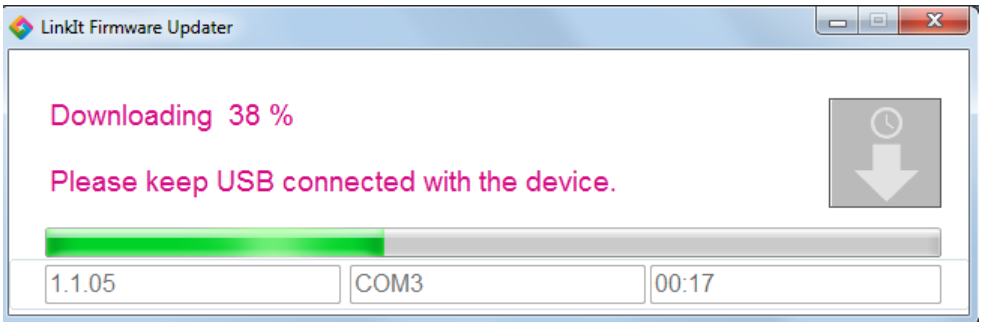

Figure 17 The board's firmware is being downloaded

6) After a moment LinkIt ONE Firmware Updater will confirm that the firmware has been updated, see Figure 18.

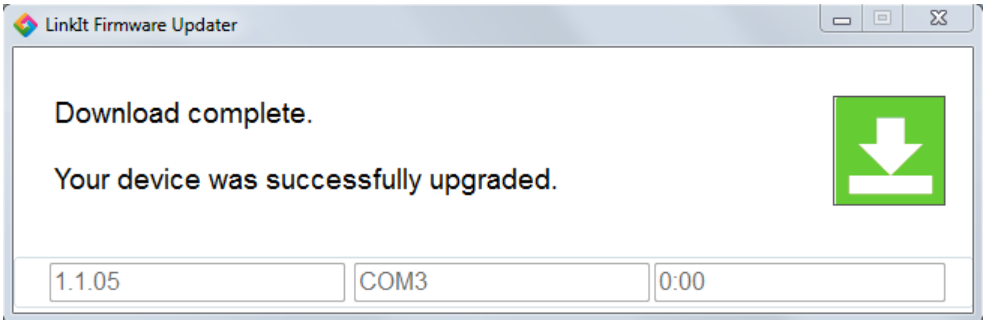

Figure 18 The firmware update is complete

7) Now return the board to normal storage mode. Then disconnect and reconnect the power to reboot it.

You should be able to upload your sketch to LinkIt ONE.

3.2 Sketches no longer uploading to LinkIt ONE

If your sketches aren't uploading to LinkIt ONE please check:

1) That LinkIt ONE is connected to your computer. To check this:
   - Open Windows Control Panel, click **System** and:
     - On Windows 7 and 8, click **Device Manager**.
     - On Windows XP, click the **Hardware** tab and then **Device Manager**.
   - In **Device Manager**, navigate to **Ports (COM & LPT)** and check that **MediaTek USB Debug Port** is displayed (see Figure 9).
2) That Arduino IDE is set to use LinkIt ONE, on the **Tools** menu point to **Board** and check that **LinkIt ONE** is selected.
3) That the port of LinkIt ONE is correctly set in the Arduino IDE. On the **Tools** menu point to **Port** and check that the correct port is selected.
4) If the sketch still won't upload, update the board's firmware as described in 3.1, "Firmware update request".

If these steps don't resolve the issue, you may find useful information in the Arduino Troubleshooting Guide or by searching the LinkIt ONE HDK discussion forum (and adding a new discussion if the issue hasn't been covered yet).

3.3 Board no longer responding

If you haven't been able to get your board working by checking the settings and updating the firmware, check the Arduino website and their discussion forums for additional advice, you can also contact technical support at Seeed Studio.

3.4 Known issues and limitations

3.4.1 Analog I/O

LinkIt ONE has 3 analog input (D0, D1, D2), whereas Arduino UNO has 6. These 3 analog input cannot be used as digital I/O, whereas those on Arduino UNO can. The reference voltage used for analog input is fixed at 5V, where on Arduino UNO there are several options.

3.4.2 Digital I/O

The driving current of LinkIt ONE is small and it can be used for signal processing only. It may have problem driving large current devices (for example LEDs, motors, relays and alike) directly. You may need a switch circuit to control these devices.

3.4.3 PWM

LinkIt ONE has 2 sets of PWMs, where Arduino UNO has 6. Also the resolution of PWM on LinkIt ONE can be up to 13-bit, where on Arduino UNO the PWM has 8-bit resolution.

3.4.4 SPI

LinkIt ONE supports SPI master mode only, whereas Arduino UNO can support both master and slave modes.

3.4.5 I2C

LinkIt ONE supports I2C master mode only, whereas Arduino UNO can support both master and slave modes.

3.4.6 SD/SPI

LinkIt ONE has built-in SD support, whereas Arduino UNO has not. To enable the SD function you have to set the jumper to the SD position. When in SD mode SPI and its corresponding pins (D13, D12 and D11) on LinkIt ONE do not work.

3.4.7 GSM/GPRS

LinkIt ONE does not support PIN-locked SIMs. If GSM/GPRS is not working, please make sure the SIM is not PIN-locked.

3.4.8 Interrupt

The interrupt pins (D2, D3) on LinkIt ONE support RISING, FALLING and CHANGE modes only. They don't support HIGH and LOW modes, unlike the Arduino UNO where the interrupt pins can support all five modes.

4 LinkIt ONE API Guide

This section provides an introduction to the LinkIt ONE APIs, full documentation of the APIs is provided in the MediaTek LinkIt ONE API reference on the MediaTek Labs website.

4.1 Digital I/O

The Digital I/O API is one of the fundamental LinkIt ONE APIs; with other APIs and function libraries being based on it (such as Advanced I/O, SPI, I2C and UART). It's used to transmit digital signals, and complements the Analog I/O API that processes analog signals.

Digital signals have two states, high (3.3V) and low (0V), indicated by static variables HIGH and LOW.

There are 16 pins on LinkIt ONE available for digital I/O; some of the pins can be shared with other APIs (see Table 3). In most cases, when you use digital I/O pins, you combine several pins together, constructing a port to communicate with a peripheral.

Board	Pins
LinkIt ONE	D0 ~ D13 SDA/SCL or D18/D19 (shared with Wire/I²C)

Table 3 Digital I/O pins are listed

Before using a digital I/O pin, you must set its pin mode. Failure to set the correct mode can result in unexpected behaviour. The default pin mode is INPUT. For details, refer to `pinMode()` in the MediaTek LinkIt ONE API reference

4.2 Advanced I/O

The Advanced I/O API combines simple logic operations based on digital I/O, for acquiring digital signals that are generated by assigned pins or are read from the voltage value of assigned pins. The logic processing is realized through software; therefore the use of pins is quite flexible; all digital I/O pins (see Table 3) can be used as advanced I/O pins.

When using a pin for advanced I/O, make sure the pin mode of that pin is set up correctly.

4.3 Analog I/O

The Analog I/O API makes use of the LinkIt ONE Analog to Digital Converter (ADC) to convert continuous analog signals into discrete digital signals read from the assigned analog pin (see Table 4), for example to read the voltage of a sensor. The process is used in reverse to convert discrete digital signals to a continuous analog signal for writing to an assigned pin.

Board	Analog input pin	Analog output pin
LinkIt ONE	A0,A1,A2	D3, D9

Table 4 The analog input and output pins

Please note the following:

- `analogWrite()` realizes analog output using PWM to drive hardware analog output.
- `analogWriteAdvance()` fine-tunes parameters for more accurate PWM control.

LinkIt ONE provides 3 analog inputs (A0 to A2) compared to 6 inputs (A0 to A5) on Arduino UNO. Although LinkIt ONE operates on 3.3V, the analog input still accepts 0 to 5V inputs and transforms them to the digital range 0 to 1023.

LinkIt ONE provides 2 PWM outputs (D3, D9) compared to 6 outputs (D3, D5, D6, D9, D10, D11) on Arduino UNO. However, the output from LinkIt ONE on D3 and D9 supports accuracy up to 13-bit. The bit accuracy effects the output frequency, with 1.6kHz offered at 13bit, 50.8kHz at 8bit and 800kHz at 4bit.

4.4 Serial

The Serial API is capable of exchanging data with PCs and other devices. Set up the same baud rate as the remote device using `begin()` to exchange data using `read()` and `write()`.

LinkIt ONE provides 2 sets of UART, the physical UART (pin 0 and 1) and USB UART. When a LinkIt ONE board is connected to a PC using a USB cable, the COM port seen in the IDE is USB UART, not the physical UART (pin 0 and 1). On the PC the port will be identified as **MTK USB Modem Port** in device Manager.

The Serial object corresponds to the USB UART. To read from and write to the physical UART (pins 0 and 1), use the Serial1 object instead, as shown in Table 5.

UART	Serial Mapping
USB UART	Serial
UART 0/1	Serial1

Table 5 Mapping of UARTs to Serial objects

4.5 Time

The Time API acquires relative time information and provides pauses.

`millis()` gets the time from when the LinkIt ONE board started in milli-seconds, while `micro()` gets the same relative time but in microseconds

`delay()` pauses the program for the set number of milliseconds, while `delayMicroseconds()` does the same but is set in microseconds.

4.6 Interrupts

The External Interrupts API sets up an Interrupt Service Routine (ISR), which is called automatically when an interrupt arrives on one of the interrupt pins (see Table 6). The interrupt service cannot preempt execution. When the next interrupt arrives, it will wait for ISR to finish the previous interrupt before its execution starts.

`interrupts()` and `noInterrupts()` enable and disable interrupts. If interrupts are disabled, care should be taken to disable them for as short a time as possible.

Board	int.0	int.1
LinkIt ONE	D2	D3

Table 6 The Interrupt pins

You cannot execute any LinkIt ONE APIs in the ISR. This is because the ISR is run in a separate thread which is of a higher priority to the one running the LinkIt ONE Sketch code. When the ISR is executed in this thread, the thread running the Sketch is suspended to wait for execution of the ISR to finish. Where LinkIt ONE APIs need to be executed as a result of the interrupt, this can be achieved by setting a global variable that is then used in the Sketch code to execute the desired APIs.

`noInterrupts()` doesn't disable all hardware interrupts, only interrupts from the pins. This enables the stable execution of other APIs, but does mean that the application may be interrupted by some events.

4.7 Math

The Math API provides four sets of maths functions:

- Math basic: Provides basic math calculations and operations.
- Bits and Bytes: Provides various bit and byte get and set functions.
- Random Numbers: Pseudo random number function library, using `randomSeed()` to assign a seed and `random()` to get the random number.
- Trigonometry: Radian based trig functions `sin()`, `cos()` and `tan()`.

4.8 Servo

The Servo API enables LinkIt ONE to control a radio control (RC) servo: An RC servo has gears and a shaft and is capable of accurate control and can position its shaft at a specific angle, normally between 0 and 180 degrees.

Only pins with PWM functionality (D3 and D9) can be used by Servo.

4.9 SPI

The SPI (Serial Peripheral Interface) API provides for data communication between the processor and various peripheral devices over a synchronous external serial port. On LinkIt ONE, SPI needs the following 3 pins to complete communication (see Table 7):

- **MISO** (Master In Slave Out): Transmits data from slave to master.
- **MOSI** (Master Out Slave In): Transmits data from master to slave.
- **SCK** (Serial Clock): Serial clock output from master, used to sync signal timing between master and slave.

Board	MOSI	MISO	SCK
LinkIt ONE	D11	D12	D13

Table 7 The SPI pins

As many slaves may be connected to one master at the same time, the **SS** (Slave Select) pin enables the Master to selects the slave to communicate with. When the SS signal of a slave is at low voltage, the master will start communicating with it.

The SPI communication protocol is very flexible; therefore before conducting data exchange using SPI, please pay extra attention to the slaves' configuration tables. Only when relevant parameter

settings comply with the demand from the slave can the master and slave start normal data communication. Before using a new slave, please answer the following questions first:

- Is the device's process for sending and receiving data "MSB first" or "LSB first"? (You set this up in `setBitOrder()`.)
- Is the clock of the device at high voltage or low voltage when it's idle? Does data exchange occur at the rising edge or falling edge of the clock cycle? (You set this up in `setDataMode()`.)
- What is the transmission speed required by the slave? (You set this up in `setClockDivider()`.)

After all the questions above are answered and the appropriate parameters set, you can use LinkIt ONE to control the device.

4.9.1 Hardware setup

The SPI and built-in SD card functions are exclusive. Before using the SPI API, make sure the settings switch is correctly set (see Figure 19). Please note that once the board is configured to SPI, calling SD APIs may result in unexpected SPI behavior.

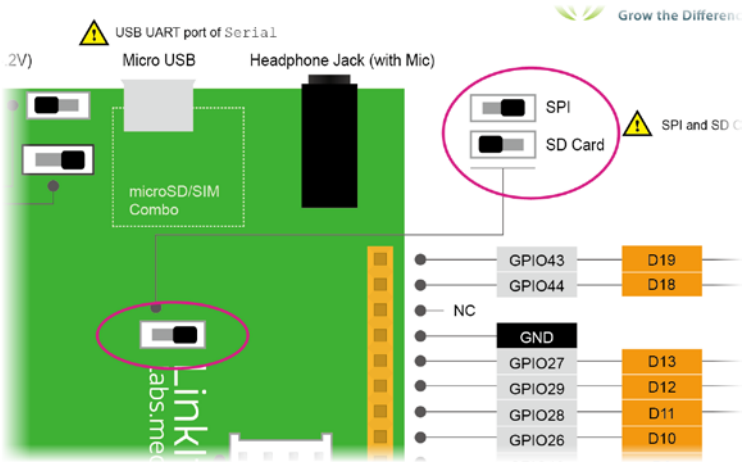

Figure 19 The SPI/SD switch

If you want to use SPI and SD functions at the same time, please use an external SD shield.

On LinkIt ONE, the processor can be used as the master only; therefore other peripheral devices can be used as slaves only.

4.10 Wire (I2C)

The Wire API is used to connect LinkIt ONE to other devices through an I2C/TWI bus using the pins shown in Table 8.

Board	[I2C/TWI] pin
LinkIt ONE	D18 Serial Data Wire (SDA) D19 Serial Clock Wire (SCL)

Table 8 The I2C pins

LinkIt ONE can join the I2C bus as a master only, not as a slave.

4.11 Stepper

The Stepper API controls unipolar and multi-polar stepper motors used in applications where precision is required, such as drawing and printing machines.

These motors receive pulse signals; stopping when there is no pulse signal and rotating when there is a pulse at a speed proportional to the pulse frequency. Immediate activation and rapid stop are its features. You can change the pulse order to change the steppers rotation direction.

4.12 GSM/GPRS

The GSM/GPRS APIs support:

- Sending and receiving Short Message Service (SMS) messages.
- Data transfer over GPRS (2G mobile network).

Using GSM/GPRS an application can connect to remote network services over TCP as a client or server. To do this first connect to the network by calling `LGPRS.attachGPRS()`. Then do the following:

- To act as a client connect to a remote TCP/IP server by creating an `LGPRSClient` object and calling `LGPRSClient.connect()`.
- To act as a server, call `LGPRSServer.available()` to detect the incoming connections. The method returns an `LGPRSClient` object representing the connection. Call the `read()` and `write()` methods to exchange data with the remote client.

LinkIt ONE has the ability to auto detect APN settings from the telecom operator, or they can be set in the application.

SIM unlock using a PIN is not supported. To use a PIN locked SIM, remove the PIN lock before inserting it onto the LinkIt ONE board.

For an example of using the GPRS APIs see 5.7, “Connecting to the web using GPRS” on page 52.

For examples of using the GSM Aps see 5.1, “Sending a Short Message Service (SMS) message” on page 31 and 5.2, “Receive a Short Message Service (SMS) message” on page 33.

4.13 Storage (SD/Flash)

LinkIt ONE provides an SD card slot and 10 MB of internal flash storage. You can manipulate these two storage areas using the `LSD` and `LFlash` classes of the Storage API.

The `LSD` class is for the LinkIt SD card slot. If you would like to use an Arduino SD shield, use the `SD` library provided by Arduino instead. As a result you can mount and use an Arduino SD shield at the same time as using the LinkIt SD card slot.

To manipulate a file or folder, use `LSD.open()` or `LFlash.open()` to get an `LFile` object then do the following:

- If the object is a file (`isDirectory()` returns `false`), use the object to read from and write to the file.
- If the object is a folder (`isDirectory()` returns `true`), use it to enumerate sub-files in this folder.

4.13.1 Hardware setup

The built-in SD card and SPI functions are exclusive. Before using the SD APIs, make sure the settings switch is correctly set (see Figure 19 on page 25). Please note that once the board is configured to SD, calling SPI APIs may result in unexpected SD behavior.

If you want to use SD and SPI at the same time, please use an external SD shield.

4.14 Bluetooth

There are two Bluetooth Profiles supported:

- Bluetooth Serial Port Profile (SPP) over Bluetooth 2.1
- Generic Attribute Profile (GATT) over Bluetooth 4.0.

4.14.1 Serial Port Profile

LinkIt ONE provides support for Bluetooth 2.1 Serial Port Profile (SPP) one-to-one connections. For more information, see the SPP page on the Bluetooth Developer Portal.

The Bluetooth API uses this feature to enable the connection of two Bluetooth devices and the exchange of data between them. The API provides classes for LinkIt ONE to act as a client (`LBTClientClass`) or server (`LBTServerClass`) in the SPP connection.

When acting as a client, LinkIt ONE will do the following:

- Scans for Bluetooth devices and connects to a designated device offering a server.
- Sends data to and receives data from the connected server.

When acting as a server, LinkIt ONE will do the following:

- Waits for the Bluetooth SPP client to connect.
- Sends data to and receives data from the connected client.

For an example of using the Bluetooth APIs, see section 5.4, “Connect an Android phone to LinkIt ONE using a Bluetooth connection” on page 39.

4.14.2 Generic Attribute Profile

LinkIt ONE also provides support for Bluetooth 4.0 Generic Attribute Profile (GATT). For more information see, the GATT page on the Bluetooth Developer Portal.

The Bluetooth API uses this feature to enable the connection of two Bluetooth devices and the exchange of data between them. The API provides classes for LinkIt ONE to act as a client (`LGATTClient`) or server (`LGATTServer`) in the GATT connection. To enable you to work with the data exchanged by these classes, the class `LGATTService` provides functions to access service details and data.

When acting as a client (GAP central device), LinkIt ONE will do the following:

- Creates a GATT client.
- Scans for Bluetooth devices providing the desired standard or custom GATT Profile and connects to a designated server device.
- Sends data to and requests data from the connected server.

When acting as a server (GAP peripheral device) LinkIt ONE will do the following:

- Creates a GATT server and defines the services it offers (as a set of services that comply with a GATT profile or a set of defined and custom services to create a custom profile).
- Waits for the Bluetooth GATT client to connect.
- Sends data to and receives data from the client.

The example application `LGATTSUart`, provided in the SDK, demonstrates how to define a service that acts as the server-side in a protocol similar to Serial Port Profile, by defining two characteristics — Tx and Rx — in the service.

4.15 GPS

LinkIt ONE provides a built-in GPS device. With the GPS API, you can get GPS data from this device.

The basic flow for controlling GPS is:

- `powerOn()`: Power on GPS
- `setMode()`: Set up work mode (optional)
- `getData()`: Query and process GPS data
- `powerOff()`: Power off GPS

The data returned from `getData()` may be GPGGA, GPGSA, GPRMC, GPVTG, GPGSV, GLGSV, GLGSA, BDGSV and BDGSA which are standard NMEA information types. Parse them to get the GPS location, time and other details.
Refer to http://www.gpsinformation.org/dale/nmea.htm for details on NMEA.

For an example of using the GPS APIs, see 5.5, “Using the Bluetooth GATT profile” on page 45.

4.16 Wi-Fi

The WiFi API provides features to enable LinkIt ONE to scan and connect to Wi-Fi APs using its built-in Wi-Fi module. With this WiFi connection ability, LinkIt ONE is able to access various resources on the Internet, including all kinds of web services.

Wi-Fi functions are provided by three classes:

- `LWiFi` provides for scanning and connecting to WiFi APs, including:
  - `begin()` to enable the Wi-Fi module
  - `connect()` to connect to an un-encrypted WiFi AP.
  - `connectWEP()` and `connectWPA()` to connect to encrypted APs.
- `LWiFiClient` provides for connecting to an Internet service using the TCP protocol, including:
  - `connect()` to open a TCP/IP connection to the web server and read the content.

  Your application can include up to 7 clients.
- `LWiFiServer` provides for setting up TCP ports and listening to remote TCP clients, including:
  - `available()` to check the incoming connections. This API returns an `LWiFiClient()` instance representing the connection.
  - `client.read()` and `client.write()` to exchange data with the remote client.

For both `LWiFiClient` and `LwiFiServer`, the `print()` and `write()` methods are available to implement Stream interfaces. It is easier to transmit ASCII strings or convert numerical values into strings with the `print()` methods. However, you may use `write()` to send low-level raw buffer content. The `print()` methods are implemented by `write()` methods.

Currently there are some limitations to the WiFi API:

- You cannot designate the group ID of the password when connecting to WEP encrypted APs.
- Querying the encryption protocol of an AP is not supported.
- There is no support for static network settings; that is static IP, DNS servers or subnet masks. Instead, DHCP is used to provide network settings.

For an example of using the WiFi APIs, see section 5.3, “Connecting to the web using Wi-Fi” on page 37.

4.17 Audio

LinkIt ONE supports playback of audio files in AMR, MP3 and AAC formats that are stored on an SD card or in the built-in flash memory. Output is through the earphone jack on a LinkIt ONE development board, see Figure 20.

Figure 20 The headphone jack on LinkIt ONE

The Audio API provides features to play, pause and stop playback, as well as adjust volume.

Playback (decoding) is handled internally by LinkIt ONE. Therefore, all functions are non-blocking, so when you call `playFile()`, it will play the audio and immediately return to your program. You can then use `getStatus()` to check the playback status.

If you try to play another audio file before the previous playback is finished, the previous playback will be stopped and the new audio file played.

4.18 Battery

The Battery API provides:

- The battery level API that returns current battery level as a percentage (0-100).
- The charging state API that returns if the battery is being charged.

4.19 DateTime

The RTC (Real Time Clock) API provides features to set and get the date and time. It provides a useful alternative to `millis()` where long time period (month and years) need to be measured. (`millis()` will overflow and reset in around 50 days.)

As long as LinkIt ONE is powered (either by power form a USB connection or by a battery) the clock will continue to tick. Where power is lost or the board rebooting, the current time can be set by either parsing the GPGGA data from `GPS.getData()` or connect to a Network Time Protocol (NTP) server over Wi-Fi.

4.20 EEPROM

LinkIt ONE provides a built-in Electrically Erasable Programmable Read-Only Memory (EEPROM). The data stored in the EEPROM is retained when the board is powered off. The EEPROM API provides read and write access to the EEPROM.

The EEPROM on LinkIt ONE can store up to 1,024 bytes.

4.21 Data type sizes

Table 9 shows the sizes of each data type available in the LinkIt ONE API along with size in the Arduino APIs.

Data type	LinkIt ONE SDK	Arduino SDK
boolean	1 byte	1 byte
byte	1 byte	1 byte
char	1 byte	1 byte
short	2 bytes	2 bytes
word	4 bytes	2 bytes
int	4 bytes	2 bytes
long	4 bytes	4 bytes
float	4 bytes	4 bytes
double	8 bytes	4 bytes

Table 9 The sizes of variables in LinkIt ONE and Arduino SDKs

5 Using the LinkIt ONE APIs

This section provides a guide to using various LinkIt ONE APIs to achieve common tasks on LinkIt ONE.

5.1 Sending a Short Message Service (SMS) message

This section describes how to configure your LinkIt ONE board and the code needed to send a Short Message Service (SMS) message.

5.1.1 Hardware setup

To prepare your LinkIt ONE development board, as shown in Figure 22, as follows:

1) Insert a standard size SIM card into the SIM holder on the rear of your board. You will need an adaptor if you're using a micro or nano SIM. Please also make sure the SIM is not PIN-locked. LinkIt ONE SDK doesn't support PIN-locked SIMs.
2) Attach a GSM/GPRS antenna to the antenna connector.

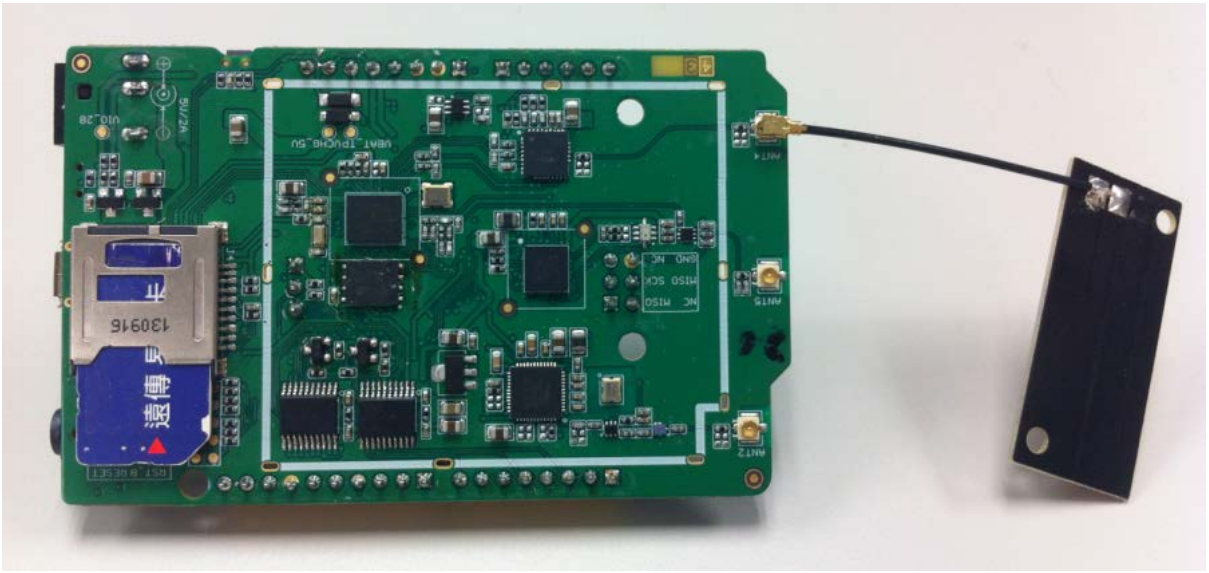

Figure 21 The LinkIt ONE development board with SIM card inserted and GSM antenna attached

5.1.2 Software setup

This section describes the steps necessary to create the code to receive your SMS.

5.1.2.1 Include the GSM library

You should include the GSM library in your code, to do this, with your Sketch active in Arduino IDE on the **Sketch** menu point to **Import Library** and click **LGSM**. You will see the GSM header is now included in your Sketch.

```
#include <LGSM.h>
```

You can now use the LSMS object to perform SMS related tasks in your Sketch. For details on the features of LSMS, please refer to LinkIt API Reference Guide.

5.1.2.2 Wait for SIM to initialize

The SIM card is a relatively slow device and can take a few seconds to initialize before you can use it. You use LSMS to check if it's ready. If it's not ready yet, wait for 1 second and try again.

In general the SIM will initialize in 5~10 seconds. If it take longer than 30 seconds, please check the SIM installation.

```
while(!LSMS.ready())
{
  delay(1000);
}
```

5.1.2.3 Send SMS out!

To send an SMS message, you'll need to provide destination numbers and the content of the message. You'll use 3 APIs: `beginSMS()`, `print()` and `endSMS()` in a simple 3 steps process:

1) Use `beginSMS()` to specify one or more destination numbers
2) Use `print()` to write data to the SMS message content
3) Use `endSMS()` to send the message

The following code snippet assume that you already have the destination numbers and the content — normally you would build these within other code in your Sketch.

```
LSMS.beginSMS("0123456789");
LSMS.print("Hello from LinkIt!");
LSMS.endSMS();
```

You should test the return value from `endSMS()` to determine if the SMS send was successful, a fail returns 0 (see 5.1.2.4, "The complete code" to see how to do this). Double check the installation of SIM and GSM antenna. If it still fail, check that the SIM is not locked and is capable of sending SMS by inserting it into a mobile phone and sending a message.

5.1.2.4 The complete code

Here is the complete Sketch code:

```
#include <LGSM.h>

void setup() {
  Serial.begin(9600);

  while(!LSMS.ready())
    delay(1000);

  Serial.println("SIM ready for work!");

  LSMS.beginSMS("0123456789");
  LSMS.print("Hello from LinkIt");
  if(LSMS.endSMS())
  {
    Serial.println("SMS is sent");
  }
  else
  {
    Serial.println("SMS is not sent");
  }
}

void loop()
{
  // do nothing
}
```

5.2 Receive a Short Message Service (SMS) message

This section describes how to configure your LinkIt ONE development board and the code needed to receive a Short Message Service (SMS) message.

5.2.1 Hardware setup

To prepare your LinkIt ONE development board, as shown in Figure 22, as follows:

1) Insert a standard size SIM card into the SIM holder on the rear of your board. You will need an adaptor if you're using a micro or nano SIM. Please also make sure the SIM is not PIN-locked. LinkIt ONE SDK doesn't support PIN-locked SIMs.

2) Attach a GSM/GPRS antenna to the antenna connector.

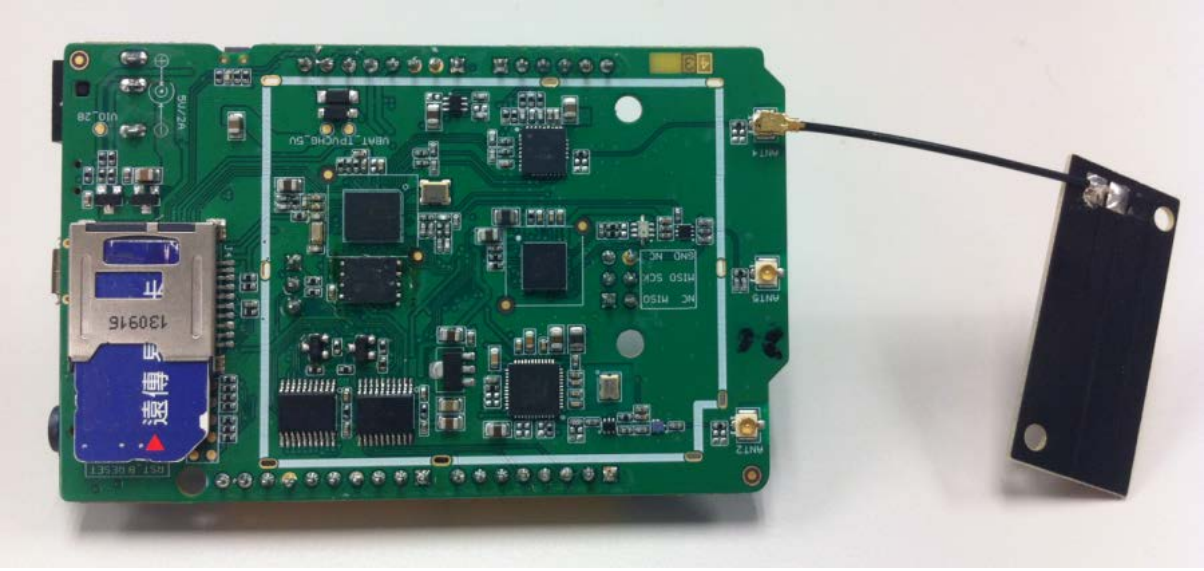

Figure 22 The LinkIt ONE development board with SIM card inserted and GSM antenna attached

5.2.2 Software setup

This section describes the steps necessary to create the code to receive your SMS.

5.2.2.1 Include the GSM library

You should include the GSM library in your code, to do this, with your Sketch active in Arduino IDE on the **Sketch** menu point to **Import Library** and click **LGSM**. You will see the GSM header is now included in your Sketch.

```
#include <LGSM.h>
```

You can now use the LSMS object to perform SMS related tasks in your Sketch. For details on the features of LSMS, please refer to LinkIt ONE API Reference Guide.

5.2.2.2 Wait for SIM to initialize

The SIM card is a relatively slow device and can take a few seconds to initialize before you can use it. You use LSMS to check if it's ready. If it's not ready yet, wait for 1 second and try again.

In general the SIM will initialize in 5~10 seconds. If it take longer than 30 seconds, please check the SIM installation.

```
while(!LSMS.ready())
{
  delay(1000);
}
```

5.2.2.3 Check if an SMS is available

The LSMS `avaiable()` API is used to check if there is SMS. You'll probably want to include the code in the `loop()` so that your board checks if a message is available regularly.

```
void loop()
{
  if(LSMS.available())
  {
    // continue to display
  }
  delay(1000);
}
```

5.2.2.4 Read the content of the SMS message

The first step is to find the source number (the number of the sender of the SMS) by providing an output buffer and size of buffer parameters to `remoteNumber()`. The length of number, and the buffer length you need, depends on the number length reported in your country, 20 characters should be enough for most cases.

The second step is to query message content using `read()`. The message content will be of a variable in length, `read()` therefore stream reads the content, one byte at a time, until it returns a negative value.

```
char buf[20];
LSMS.remoteNumber(buf, 20); // number is stored into buf

int c;
while(true)
{
  c = LSMS.read();          // message content (one byte at a time)
  if(c < 0)
    break;                  // enf of message content
}
```

5.2.2.5 Delete an SMS

After the number and content has been read, the message should be removed so that the next message can be retrieved. To delete the SMS last checked by `available()`, simply use `flush()`.

```
LSMS.flush()
```

5.2.2.6 The Complete Code

Here is the complete Sketch code:

```
#include <LGSM.h>

void setup() {
  Serial.begin(9600);

  while(!LSMS.ready())
    delay(1000);

  Serial.println("SIM ready for work!");

}

void loop()
{
  char buf[20];
  int v;
  if(LSMS.available())            // Check if there is new SMS
  {
    Serial.println("There is new message.");

    LSMS.remoteNumber(buf, 20);    // display Number part
    Serial.print("Number:");
    Serial.println(buf);

    Serial.print("Content:");      // display Content part
    while(true)
    {
      v = LSMS.read();
      if(v < 0)
        break;
      Serial.print((char)v);
    }
    Serial.println();

    LSMS.flush();                  // delete message
  }

  delay(1000);
}
```

5.3 Connecting to the web using Wi-Fi

This section describes how to configure your LinkIt ONE development board and the code needed to connect to a Wi-Fi access points (AP) and retrieve the content of a web page.

5.3.1 Hardware setup

To prepare your LinkIt ONE development board, as shown in Figure 23, by attaching a Wi-Fi antenna to the antenna connector.

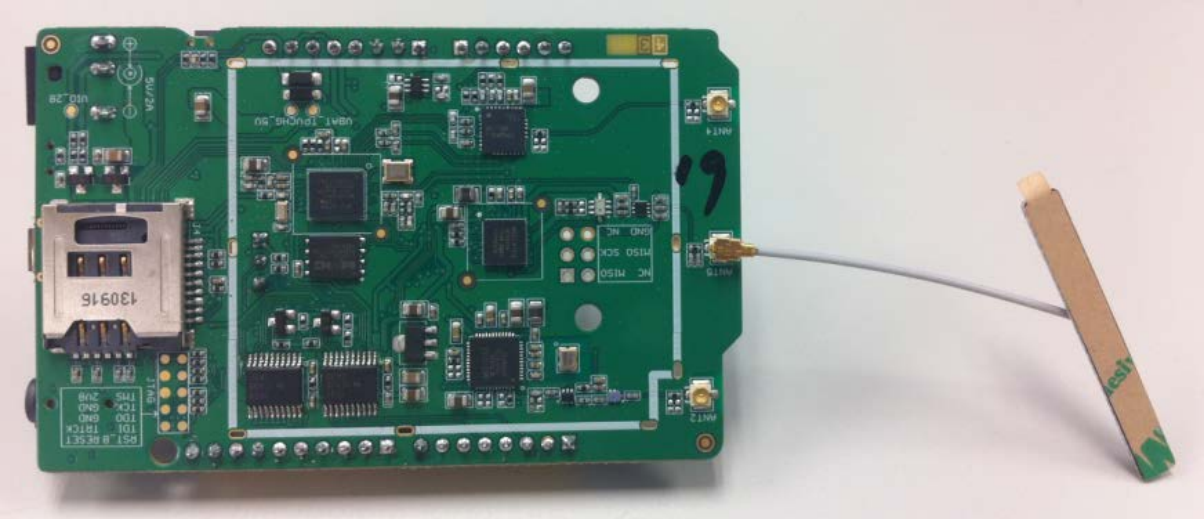

Figure 23 The LinkIt ONE development board with a Wi-Fi antenna attached

5.3.2 Software setup

This section describes the steps necessary to create the code to setup a Wi-Fi connection as well as retrieve web content.

5.3.2.1 Include the Wi-Fi library

You should include the Wi-Fi library in your code, to do this, with your Sketch active in Arduino IDE on the **Sketch** menu point to **Import Library** and click **LWiFi**. You'll see the Wi-Fi headers are now included in your Sketch. In this guide LinkIt ONE is used as the Wi-Fi client, so keep the first 2 headers and remove the others.

```
#include <LWiFi.h>
#include <LWiFiClient.h>
```

You can now use the `LWiFi` and `LWiFiClient` object to perform Wi-Fi related tasks in your Sketch. For details on the features of `LWifi` and `LWiFiClient`, please refer to LinkIt ONE API Reference Guide.

5.3.2.2 Connect to Wi-Fi access point (AP)

The first step is to enable the Wi-Fi API. To do this, use `LWiFi.begin()`. There are several ways to connect to a Wi-Fi AP. You use `LWiFi.connect()` if your AP isn't encrypted. If it's encrypted, please check which encryption it's using. LinkIt ONE supports WEP and WPA encryption. For WEP encryption, use `LWiFi.connectWEP()`. For WPA encryption, use `LWiFi.connectWPA()`.

```
#define WIFI_AP "Name_of_your_AP"
#define WIFI_PWD "Password_of_your_AP"

LWiFi.begin();
LWiFi.connect(WIFI_AP);               // if the AP is not encrypted
LWiFi.connectWEP(WIFI_AP, WIFI_PWD);  // if the AP uses WEP encryption
LWiFi.connectWPA(WIFI_AP, WIFI_PWD);  // if the AP uses WPA encryption
```

If the connection fails the value returned from `connect()` is negative. If this happens, make sure the antenna is attached correctly and LinkIt ONE is in range of the target AP.

5.3.2.3 Connect to a web site

The second step is to connect the web site. This is done using the `LWiFiClient` object. Once connected to the Wi-Fi AP, a maximum of 7 connection can be made at the same time. In this example only 1 connection is required. The first parameter of `connect()` is the URL to connect to, and the second parameter is the port; HTTP uses port 80.

```
#define SITE_URL "www.mediatek.com"

LWiFiClient c;
c.connect(SITE_URL, 80);
```

5.3.2.4 Send HTTP request

After a connection has been made, it acts like a stream where you can read from or write to it. To retrieve the web content, you'll have to send a HTTP GET request, as shown in the following code snippet:

```
c.println("GET / HTTP/1.1");
c.println("Host: " SITE_URL);
c.println("Connection: close");
c.println();
```

5.3.2.5 Get the web content

If everything is correct, the remote web server will respond to your GET request and start to return the content. You can then read the data from connection object as follows:

```
int v;
while(c.available())
{
  v = c.read();      // return one byte at a time
  if(v < 0)
    break;           // no more data
}
```

5.3.2.6 The complete code

Here is the complete Sketch code:

In this code the retrieved data is printed to a serial port with its baud rate set to 9600.

```
#include <LWiFi.h>
#include <LWiFiClient.h>

#define SITE_URL "www.mediatek.com"
#define WIFI_AP "Name_of_your_AP"          // replace with your setting
#define WIFI_PWD "Password_of_your_AP"     // replace with your setting

LWiFiClient c;

void setup() {

  Serial.begin(9600);
  LWiFi.begin();

  Serial.println();
  Serial.print("Connecting to AP...");
  if(LWiFi.connectWEP(WIFI_AP, WIFI_PASSWORD) < 0)
  {
    Serial.println("FAIL!");
    return;
  }
  Serial.println("ok");

  Serial.print("Connecting to site...");
  if(!c.connect(SITE_URL, 80))
  {
    Serial.println("FAIL!");
    return;
  }
  Serial.println("ok");

  Serial.println("send HTTP GET request");
  c.println("GET / HTTP/1.1");
  c.println("Host: " SITE_URL);
  c.println("Connection: close");
  c.println();
}

void loop() {
  int v;
  while(c.available())
  {
    v = c.read();
    if(v < 0)
      break;
    Serial.print((char)v);
  }
  delay(100);
}
```

5.4 Connect an Android phone to LinkIt ONE using a Bluetooth connection

This section describes how to configure your LinkIt ONE development board and the code needed to exchange data with an Android phone using the Bluetooth SPP profile. In this guide, the Android phone act as master and LinkIt ONE the slave.

5.4.1 Hardware setup

Prepare your LinkIt ONE development board by attaching a Wi-Fi antenna to the antenna connector, as shown in Figure 24. (Wi-Fi and Bluetooth share the same antenna.)

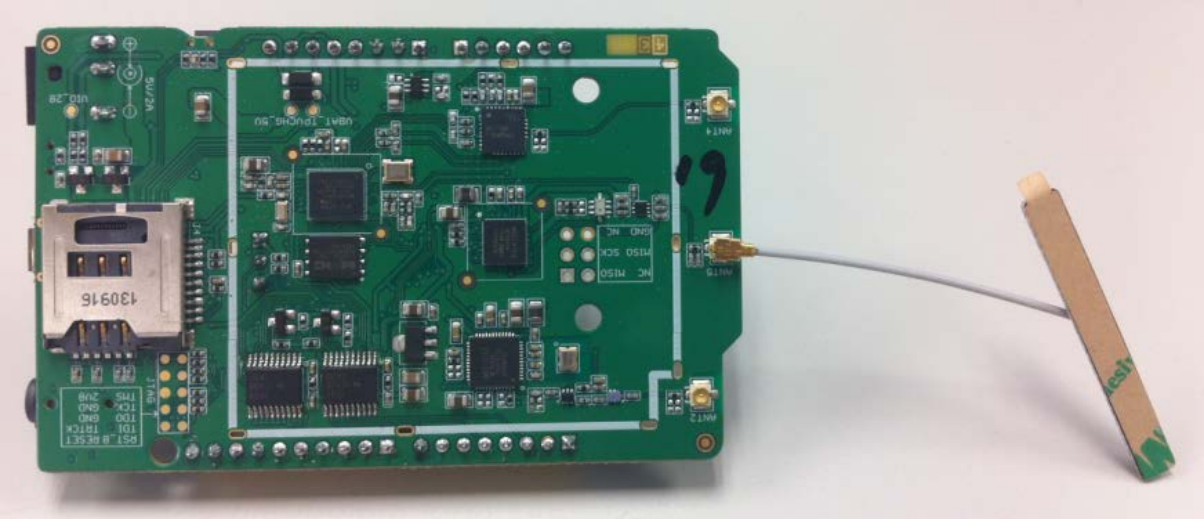

Figure 24 The LinkIt ONE development board with a Wi-Fi/Bluetooth antenna attached

5.4.2 Software setup (LinkIt ONE Side)

This section describes the steps necessary to create the code to setup a Bluetooth server and enable an Android phone to connect to LinkIt ONE.

5.4.2.1 Include the Bluetooth library

You should include the Bluetooth (BT) library in your code, to do this, with your Sketch active in Arduino IDE on the **Sketch** menu point to **Import Library** and click **LBT**. You'll see the BT headers are now included in your Sketch. In this guide LinkIt ONE is used as BT server, so only the `LBT` and `LBTServer` headers are needed, the other can be removed.

```
#include <LBT.h>
#include <LBTServer.h>
```

You can now use the `LBTServer` object to perform BT related tasks in your Sketch. For details on the features of `LBTServer`, please refer to LinkIt ONE API Reference Guide.

5.4.2.2 Start the Bluetooth Server

For a Bluetooth client to identify which devices are available to connect with, you need to specify a name during the starting process.

```
LBTServer.begin((uint8_t*)"My_BTServer")
```

5.4.2.3 Wait for the client connection

To enable a client connection, you have to ask the server to `accept()`, whose parameter is a timeout value in second. In this guide, LinkIt ONE will continue to wait for client connection forever. When a connection is made. LinkIt ONE switches to data exchange mode.

```
void loop() {
  if(LBTServer.connected())
  {
    // There is active connection
  }
  else
  {
    // Wait 5 secs and retry forever
    LBTServer.accept(5);
  }
}
```

5.4.2.4 Exchange data with the connected client

After the connection is made, you use `read()` to read data from client and `write()` to write data to client. In this guide, LinkIt ONE will simply act as an 'echo' machine: it will write back everything read from client.

```
uint8_t buf[64];
int bytesRead;

while(true)
{
  bytesRead = LBTServer.read(buf, 64); // read from client
  if(!bytesRead)
    break;
  LBTServer.write(buf, bytesRead);     // write the same data back to client
}
```

5.4.2.5 The complete code

Here is the complete Sketch code:

```
#include <LBT.h>
#include <LBTServer.h>

void setup() {
  Serial.begin(9600);

  if(!LBTServer.begin((uint8_t*)"My_BTServer"))
  {
    Serial.println("Fail to start BT.");
    return;
  }

  Serial.println("BT server is started.");
}

void loop() {
  uint8_t buf[64];
  int bytesRead;

  if(LBTServer.connected())
  {
    // echo back all received data
    while(true)
    {
      bytesRead = LBTServer.readBytes(buf, 64);
      if(!bytesRead)
        break;
      Serial.write(buf, bytesRead);
      LBTServer.write(buf, bytesRead);
    }
    delay(100);
  }
  else
  {
    LBTServer.accept(5);
  }
}
```

5.4.3 Software setup (Android Side)

This section describes the steps necessary to create the Android app that can connect and talk to LinkIt ONE.

5.4.3.1 Get Bluetooth Chat source code

Android SDK includes an example app called “BluetoothChat”, its design to allow 2 Android devices to talk to each other over a Bluetooth connection. With little modification it can also talk to LinkIt ONE devices.

The source code is located in the Android SDK code examples folder. In a standard Android SDK installation the example can be found in `C:\Program files\Android\android-sdk\samples\android-16\BluetoothChat`.

5.4.3.2 Modify the example to talk to LinkIt ONE

To enable the app to talks with a LinkIt ONE development board and our LinkIt ONE code, you need to make 2 minor modification:

1) In `AndroidManifest.xml` modify
   - `<uses-sdk … />`
   - to
   - `<uses-sdk android:maxSdkVersion="17" android:targetSdkVersion="11" android:minSdkVersion="11" />`
2) **In** `src/com/example/android/BluetoothChat/BluetoothChatService.java` modify the UUID setting of the SPP profile as follows:

```
// Unique UUID for this application
private static final UUID MY_UUID_SECURE =
    UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
private static final UUID MY_UUID_INSECURE =
    UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
```

5.4.3.3 Build the app

Now build the app. For information on how to build Android example code, please refer to the Android Developer web site.

5.4.3.4 Test the Bluetooth communication

To test the Bluetooth communication, start your Sketch on LinkIt ONE and install and run the example app on your Android phone, then:

1) The example app tap the left-most find button in the menu, as shown in Figure 25. This will list all the discoverable Bluetooth devices and should including one named **My_BTServer** the Bluetooth server name set up on your LinkIt ONE.

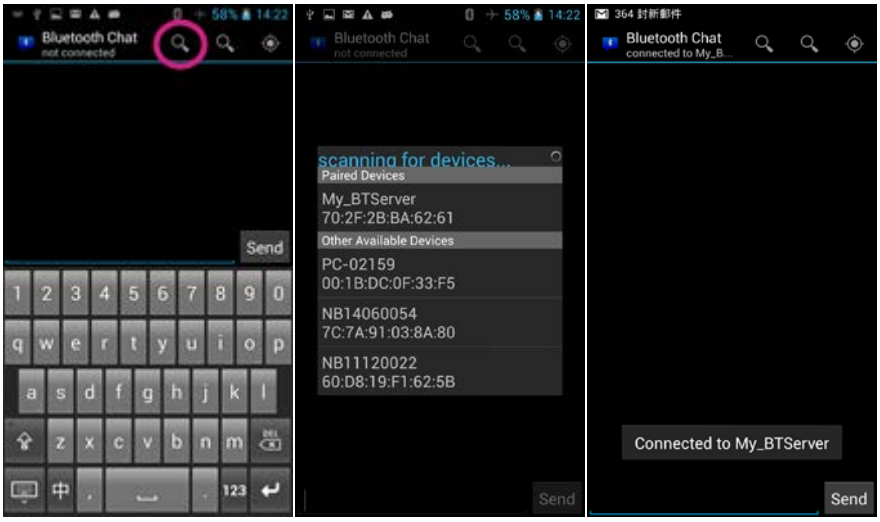

Figure 25 Connecting to the LinkIt ONE Bluetooth server

2) Select **My_BTServer** and **Connected to My_BTServer** will display, see Figure 26.

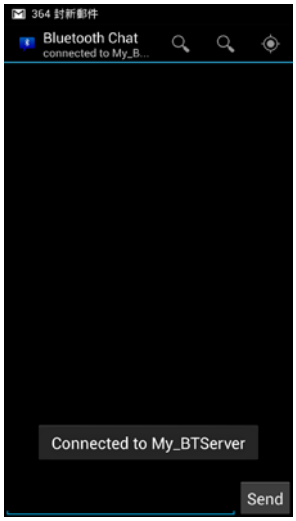

Figure 26 Connection to the LinkIt ONE Bluetooth server confirmed

3) Now type the content to be sent to LinkIt ONE and you should see it return back from LinkIt ONE, as shown in Figure 27.

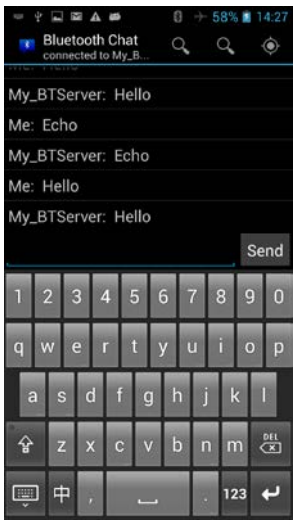

Figure 27 The entered text echoed back from LinkIt ONE

5.5 Using the Bluetooth GATT profile

Depending on your needs, you can either program LinkIt ONE as a central device that connects to other Bluetooth 4.0 devices and accesses their profiles and services, or program LinkIt ONE as a peripheral device that provides profiles and services for other devices to use.

5.5.1 Hardware Setup

Prepare your LinkIt ONE development board by attaching a Wi-Fi antenna to the antenna connector, as shown in Figure 24. (Wi-Fi and Bluetooth share the same antenna.)

5.5.2 Implementing a General Attribute Profile (GATT) server

To define the LinkIt ONE development board as a GATT server (GAP peripheral device), you need to define a GATT server profile. You do this by calling the `begin()` method of an `LGATTServer` object.

A GATT profile consists of one or more services, see the GATT page on the Bluetooth Developer Portal for more information. In this example a profile with two services is created. The `begin()` method requires you to pass the number of services in your role, and instances of sub-class of `LGATTService` which defines the actual behavior of all the services in your GATT profile, as shown in the code below.

```
#include <LGATTServer.h>

void setup() {
  LGATTServer.begin(2, &myServiceA, &myServiceB);
}
```

A GATT service should handle requests from GATT clients, which is read and write requests to the characteristics of the service. These requests are then passed to the service instances as event callbacks, such as `LGATTService.onRead()`. Therefore, a GATT server should periodically check for incoming requests and dispatch the events to service objects. To handle the incoming events, call `LGATTServer.handleEvents()` in the `loop()` function of your Arduino sketch.

```
void loop() {
  LGATTServer.handleEvents();
}
```

To define services, inherit the LGATTService class, and implement methods that handle incoming events. You don't have to invoke these methods directly, instead these methods are invoked by the GATT library appropriately when LGATTServer.handleEvents() is called. The following code shows how this is done by defining a class that inherits LGATTService class and overrides the onLoadService method:

```
class LGATTSample : public LGATTService {
public:
    virtual LGATTServiceInfo *onLoadService(int32_t index);
};
static LGATTServiceInfo MY_SERVICE_DEFINITION [] = {
    {TYPE_SERVICE, "6e400001-b5a3-f393-e0a9-e50e24dcca9e", TRUE, 0, 0, 0},
    {TYPE_CHARACTERISTIC, "6e400002-b5a3-f393-e0a9-e50e24dcca9e", FALSE,
     VM_GATT_CHAR_PROP_WRITE, VM_GATT_PERM_WRITE, 0},
    {TYPE_CHARACTERISTIC, "6e400003-b5a3-f393-e0a9-e50e24dcca9e", FALSE,
     VM_GATT_CHAR_PROP_NOTIFY | VM_GATT_CHAR_PROP_INDICATE, VM_GATT_PERM_READ, 0},
    {TYPE_DESCRIPTOR, "00002902-0000-1000-8000-00805f9b34fb", FALSE,
     VM_GATT_CHAR_PROP_NOTIFY, VM_GATT_PERM_READ | VM_GATT_PERM_WRITE, 0},
    {TYPE_END, 0, 0, 0, 0, 0}
};
LGATTServiceInfo* LGATTSample::onLoadService(int32_t index){
    return MY_SERVICE_DEFINITION;
}
```

onLoadService() is called before any other method in the class. As you can see in the code above, this method should return an array of ard_gatts_service_decl_struct that describes information about the service and all its characteristics and descriptors. Note that this structure must remain unchanged until LGATTServer.end() is called. Therefore, you should avoid declaring a data structure array in local scope. Either define it as a global variable, or allocate it on the heap. The returned array must start with an element of type TYPE_SERVICE, and has its own UUID. Refer to API documentation for other fields in the element.

```
    {TYPE_SERVICE, "6e400001-b5a3-f393-e0a9-e50e24dcca9e", TRUE, 0, 0, 0},
```

The array continues with elements for one or more characteristic entries, followed by zero or more descriptor entries. Again, each entry has its own UUID and different read/write permissions and properties.

```
    {TYPE_CHARACTERISTIC, "6e400002-b5a3-f393-e0a9-e50e24dcca9e", FALSE,
VM_GATT_CHAR_PROP_WRITE, VM_GATT_PERM_WRITE, 0},
    {TYPE_CHARACTERISTIC, "6e400003-b5a3-f393-e0a9-e50e24dcca9e", FALSE,
VM_GATT_CHAR_PROP_NOTIFY | VM_GATT_CHAR_PROP_INDICATE, VM_GATT_PERM_READ, 0},
    {TYPE_DESCRIPTOR, "00002902-0000-1000-8000-00805f9b34fb", FALSE,
VM_GATT_CHAR_PROP_NOTIFY, VM_GATT_PERM_READ | VM_GATT_PERM_WRITE, 0},
```

To mark the end of the array, place an element of TYPE_END:

```
    {TYPE_END, 0, 0, 0, 0, 0}
```

The LGATTServer class will then parse this array and generate corresponding internal resources. Then, it invokes the following methods to notify you about the handles to these internal resources. You should store these handles, as they are needed when attributes are being read or written by a connected central device.

- onCharacteristicAdded(): This method is called after onLoadService() and passes in the handle for each characteristic defined in the service structure. These handles should be used when calling the send(), onRead() or onWrite() methods.

- onDescriptorAdded(): similar to onCharacteristicAdded(), but called for each descriptor element in the array returned by onLoadService().

When LGATTServer.begin() returns, and onLoadService(), onCharacteristicAdded() and onDescriptorAdded() have all been invoked, LinkIt ONE starts listening for central devices that conform to your defined device profile. If there is an incoming connection, onConection() is called with the data.connected field set to true. If the central device disconnects, onConection() is called again with the data.connected field set to false.

If the central device is connected and tries to read a certain characteristic or descriptor, onRead() is invoked for each read request from the connected central device. You should call the ackOK() method inside the implementation of onRead() to transmit requested attribute value back to the central device. An acknowledgement (ACK) will be sent first, followed by values of the characteristic or descriptor, as shown below:

```
boolean LGATTSample::onRead(LGATTReadRequest &request){
        LGATTAttributeValue value = {0};
        const char *str = "value string";
        memcpy(value.value, str, strlen(str));
        value.len = strlen(str);
        request.ackOK(value);
}
```

Note that you should call request.ackOk() at most once for each request.

onWrite() works in a similar way, but after sending ACK back to the central device, instead of sending data, you should update your device state based on the value sent by the central device.

```
boolean LGATTSample::onWrite(LGATTWriteRequest &data){
      // if need to rsp to central.
      if (data.need_rsp){
            data.ackOK(); // Send ACK
      }
      // Update device internal status by the written data according.
      // Here we simply print the length of received data.
      Serial.print("Received data length:");
      Serial.println(data.value.len);
      return true;
}
```

One thing to notice is that there is a limit in LinkIt ONE such that you can only read and write 20 bytes of data in each transaction. If your service defines attributes that have values longer than 20 bytes, the client has to read or write multiple times with the offset field in the read and write request denoting which part of the attribute value to be read or written.

5.5.3 Implementing a General Attribute Profile (GATT) client

Implementing a GATT client (central device) involves discovering nearby peripherals that act as GATT servers and enumerate their profiles. Therefore, accessing a GATT server profile involves following steps:

- Scan for nearby GATT peripheral devices and their profiles.
- Connect to the desired peripheral device.
- Read or write values of characteristics of the services provided by the connect device.

The following sample shows how to scan for nearby GATT devices. First enable the GATT client, instantiate an LGATTClient object using LGATTClient.begin() and pass a UUID that identifies your application. This UUID can be any UUID generated with a suitable UUID generator. The registration may take some time to complete. Then call scan() to scan nearby GATT devices, and retrieve their

information including the RSSI value that could be used as a relative indication of distances between the two devices.

```
#include <LGATTClient.h>
LGATTUUID appUUID("AFA5B1C5-2B7B-471B-BD4C-7A92DE9D6DBD");
LGATTClient client;
void setup() {
    client.begin(appUUID);
    int numberOfDevices = client.scan(3);
    LGATTDeviceInfo info = {0};
    for(int i = 0; i < numberOfDevices; ++i){
        client.getScanResult(i, info);
    }
}
```

Once the scan has identified devices, you can then connect to one and discover the services it provides. For example, if you want to connect to a heart rate band, using the Heart Rate Profile (HRP), it must provide two services, Heart Rate Service and Device Info Service. Therefore, to detect if a device supports HRP, you call `LGATTClient.getServiceCount()` and then `getServiceInfo()` for each index to see if the UUID for HRS and DIS appear in the resulting list. If they both appear in the list, the device supports HRP and you can start querying for the corresponding characteristics.

The example below shows how to get all the services supported by a device.

```
client.connect(info.bd_addr);
int numberofServices = client.getServiceCount();
// all services uuid supported by this device
for(int i = 0; i < numberofServices; ++i){
    LGATTUUID serviceUUID;
    boolean isPrimary = false;
    client.getServiceInfo(i, serviceUUID, isPrimary);
    Serial.println(serviceUUID);
}
client.disconnect(info.bd_addr);
```

Once you've confirmed that the connected device supports the profiles that you're looking for, you can start reading attributes with `LGATTClient.readCharacteristic()`:

```
// read characteristic
LGATTUUID characteristicUUID = 0x2A37;
LGATTAttributeValue attrValue;
boolean isPrimary;
if(client.readCharacteristic(serviceUUID, isPrimary, characteristicUUID, attrValue)){
    Serial.print("characteristic value:");
    Serial.print((char*)attrValue.value);
    Serial.println();
}
```

And writing attributes with `LGATTClient.writeCharacteristic()`:

```
// write characteristic
LGATTAttributeValue attrValue;
char szbuf[] = "value to write";
memset(&attrValue, 0, sizeof(attrValue));
memcpy(attrValue.value, szbuf, strlen(szbuf));
LGATTUUID writeUUID = 0x2A39;
attrValue.len = strlen(szbuf);
if (client.writeCharacteristic(serviceUUID, isPrimary, writeUUID, attrValue)){
    Serial.println("Attribute written");
}
```

5.6 Using GPS

This section describes how to configure your LinkIt ONE development board and the code needed to use the LinkIt ONE built-in GPS functionality. The guide illustrate how to power on GPS and wait until it obtains a location fixed. After the location is fixed, retrieve the latitude/longitude information and how many satellites are visible.

5.6.1 Hardware setup

Prepare your LinkIt ONE development board, by attaching a GPS antenna to the antenna connector, as shown in Figure 28.

Figure 28 The LinkIt ONE development board with a GPS antenna attached

5.6.2 Software setup

This section describes the steps necessary to create the code to setup and use the GPS module.

5.6.2.1 Include the GPS library

You should include the GPS library in your code, to do this, with your Sketch active in Arduino IDE on the **Sketch** menu point to **Import Library** and click **GPS**. You'll see the GPS headers are now included in your Sketch.

```
#include <LGPS.h>
```

You can now use the LGPS object to perform GPS related tasks in your Sketch. For details on the features of LGPS, please refer to LinkIt API Reference Guide.

5.6.2.2 Power on the GPS module

Power on the GPS as follows:

```
LGPS.powerOn();
```

5.6.2.3 Retrieve GPS data

GPS data can be retrieved using `getData()`. The parameter is a structure containing all the GPS information for the board's location. The GPGGA element of this structure has the information needed by your Sketch: whether the location is locked, how many satellites are visible and the latitude and longitude.

Information on the format of GPGGA file can be found on the NMEA web site, but for this guide simply note that the details needed are separated by a comma.

To parse the data from the GPGGS file use `nextToken()` that parses the input string and stores the resulting tokens in a buffer. In the parsed data the first token is `$GPGGA`, the 2nd UTC time, the 3rd to 6th tokens are latitude and longitude, the 7th token indicates whether a GPS fix was achieved and the 8th token is the number of satellites visible.

```
gpsSentenceInfoStruct info;
LGPS.getData(&info);
printGPGGA((char*)info.GPGGA);

void printGPGGA(const char* str)
{
  char latitude[20];
  char longitude[20];
  char buf[20];
  const char* p = str;

  p = nextToken(p, 0);      // GGA
  p = nextToken(p, 0);      // Time
  p = nextToken(p, latitude);  // Latitude
  p = nextToken(p, 0);      // N
  p = nextToken(p, longitude); // Longitude
  p = nextToken(p, 0);      // E
  p = nextToken(p, buf);       // fix quality

  if(buf[0] == '1')
  {
    // GPS fix
    p = nextToken(p, buf);     // number of satellites
    Serial.print("GPS is fixed:");
    Serial.print(atoi(buf));
    Serial.println(" satellite(s) found!");
    Serial.print("Latitude:");
    Serial.println(latitude);
    Serial.print("Longitude:");
    Serial.println(longitude);
  }
  else
  {
    Serial.println("GPS is not fixed yet.");
  }
}

const char *nextToken(const char* src, char* buf)
{
  int i = 0;
  while(src[i] != 0 && src[i] != ',')
    i++;
```

```
  if(buf)
  {
    strncpy(buf, src, i);
    buf[i] = 0;
  }

  if(src[i])
    i++;
  return src+i;
}
```

5.6.2.4 The complete code

Here is the complete Sketch code:

```
#include <LGPS.h>

gpsSentenceInfoStruct info;

const char *nextToken(const char* src, char* buf)
{
  int i = 0;
  while(src[i] != 0 && src[i] != ',')
    i++;

  if(buf)
  {
    strncpy(buf, src, i);
    buf[i] = 0;
  }

  if(src[i])
    i++;
  return src+i;
}

void printGPGGA(const char* str)
{
  char latitude[20];
  char longitude[20];
  char buf[20];
  const char* p = str;

  p = nextToken(p, 0);       // GGA
  p = nextToken(p, 0);       // Time
  p = nextToken(p, latitude);  // Latitude
  p = nextToken(p, 0);       // N
  p = nextToken(p, longitude); // Longitude
  p = nextToken(p, 0);       // E
  p = nextToken(p, buf);       // fix quality

  if(buf[0] == '1')
  {
    // GPS fix
    p = nextToken(p, buf);     // number of satellites
    Serial.print("GPS is fixed:");
    Serial.print(atoi(buf));
    Serial.println(" satellite(s) found!");
    Serial.print("Latitude:");
    Serial.println(latitude);
    Serial.print("Longitude:");
    Serial.println(longitude);
  }
```

```
    else
    {
      Serial.println("GPS is not fixed yet.");
    }
  }

  void setup() {
    Serial.begin(9600);
    LGPS.powerOn();
  }

  void loop() {
    LGPS.getData(&info);
    printGPGGA((char*)info.GPGGA);
    delay(1000);
  }
```

5.7 Connecting to the web using GPRS

This section describes how to configure your LinkIt ONE development board and the code needed to retrieve the content of a web page over a GPRS connection.

5.7.1 Hardware setup

To prepare your LinkIt ONE development board, as shown in Figure 29, as follows:

1) Insert a standard size SIM card into the SIM holder on the rear of your board. You'll need an adaptor if you're using a micro or nano SIM. Please also make sure the SIM is not PIN-locked. LinkIt SDK doesn't support PIN-locked SIMs.
2) Attach a GSM/GPRS antenna to the antenna connector.

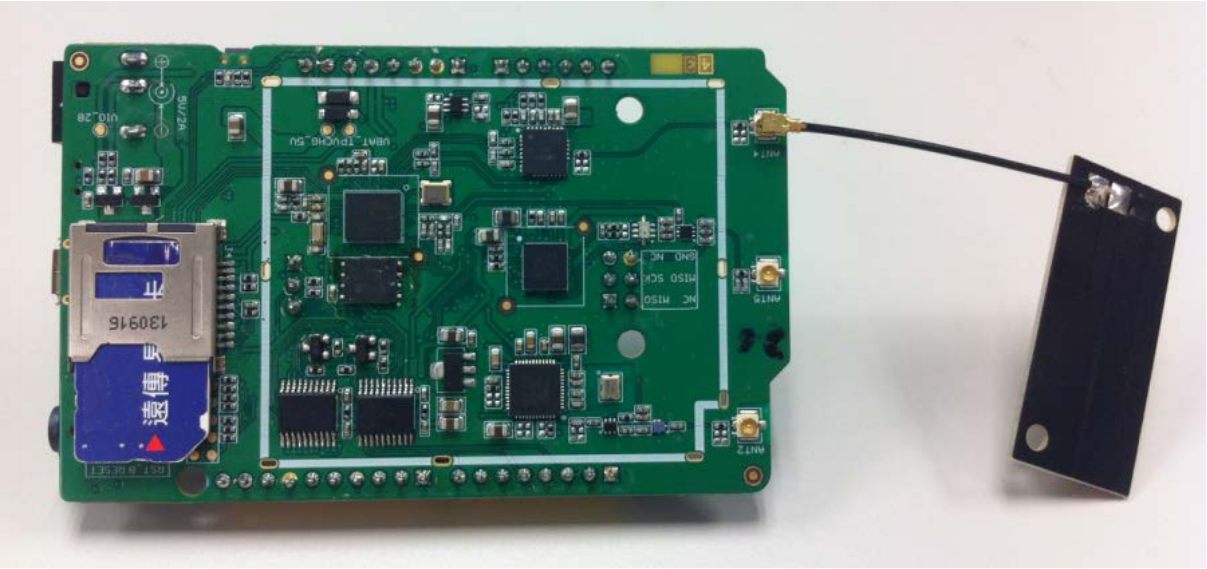

Figure 29 The LinkIt ONE development board with SIM card inserted and GSM antenna attached

5.7.2 Software setup

This section describes the steps necessary to create the code to retrieve web content over a GPRS connection.

5.7.2.1 Include the GPRS library

You should include the GPRS library in your code, to do this, with your Sketch active in Arduino IDE on the **Sketch** menu point to **Import Library** and click **LGPRS**. You'll see the GPRS headers are now included in your Sketch. In this guide LinkIt is used as the GPRS client, so keep the first 2 headers and remove the others.

```
#include <LGPRS.h>
#include <LGPRSClient.h>
```

You can now use the `LGPRS` and `LGPRSClient` object to perform GPRS related tasks in your Sketch. For details on the features of `LGPRS` and `LGPRSClient`, please refer to LinkIt ONE API Reference Guide.

5.7.2.2 Wait for GPRS to initialize

Wait for GPRS module to initialize. When it is ready, `attachGPRS()` should return non-zero.

```
while(!LGPRS.attachGPRS())
{
  delay(1000);
}
```

5.7.2.3 Connect to a web site

The second step is to connect the web site. This is done using the `LGPRSClient` object. The first parameter of `connect()` is the URL to connect to, and the second parameter is the port; HTTP uses port 80.

```
#define SITE_URL "www.mediatek.com"

LGPRSClient client;
client.connect(SITE_URL, 80);
```

5.7.2.4 Send the HTTP request

After a connection has been made, it acts like a stream where you can read from or write to it. To retrieve the web content, you'll have to send a HTTP GET request, as shown in the following code snippet:

```
client.println("GET / HTTP/1.1");
client.println("Host: " SITE_URL ":80");
client.println();
```

5.7.2.5 Get the web content

If everything is correct, the remote web server will respond to your GET request and start to return the content. You can then read the data from connection object as follows:

```
int v;
while(c.available())
{
  v = c.read();      // return one byte at a time
  if(v < 0)
    break;           // no more data
}
```

5.7.2.6 The complete code

Here is the complete Sketch code:

In this code the retrieved data is printed to a serial port with its baud rate set to 9600.

```
#include <LGPRS.h>
#include <LGPRSClient.h>

#define SITE_URL "www.mediatek.com"

LGPRSClient client;

void setup()
{
  Serial.begin(9600);

  while(!LGPRS.attachGPRS())
  {
    Serial.println("wait for SIM card ready");
    delay(1000);
  }

  Serial.print("Connecting to : " SITE_URL "...");
  if(!client.connect(SITE_URL, 80))
  {
    Serial.println("FAIL!");
    return;
  }
  Serial.println("done");

  Serial.print("Sending GET request...");
  client.println("GET / HTTP/1.1");
  client.println("Host: " SITE_URL ":80");
  client.println();
  Serial.println("done");
}

void loop()
{
  int v;
  while(client.available())
  {
    v = client.read();
    if (v < 0)
      break;

    Serial.write(v);
  }
  delay(500);
}
```