EVB_M1 Technical Manual

Contents

1 Hardware

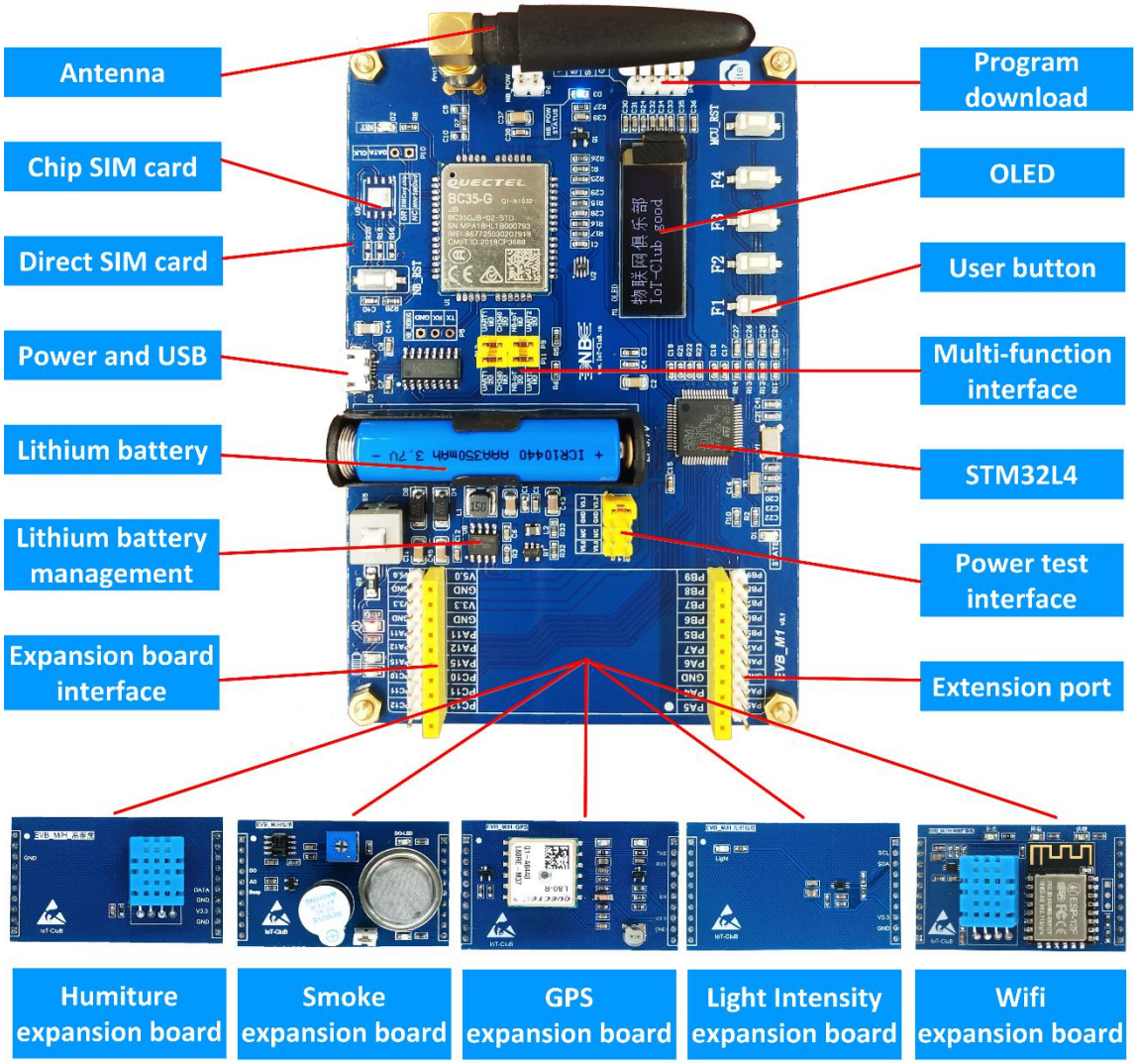
1.1 Introduction to the Development Board

1.1.1 Resources of the EVB_M1 Development Board

Since its launch in 2017, the EVB_M1 development board has been widely recognized by customers. The latest EVB_M1 3.1 development board is improved based on the original development board according to customers' feedbacks. The following describes the EVB_M1 3.1 development board.

Figure 1-1 shows the resources provided by the EVB_M1 V3.1 development board.

Figure 1-1 EVB_M1 resources



As shown in Figure 1-1, the EVB_M1 development board has abundant resources and is designed with reference to different application scenarios. NB-IoT solutions can be verified on the development board. With various interfaces and functional modules expanded, this development board provides outstanding performance.

The development board is 8 cm wide and 12 cm long. It is user-friendly designed, takes various application scenarios of NB-IoT into account, and undergoes several rounds of optimization based on suggestions from Huawei engineers and customers.

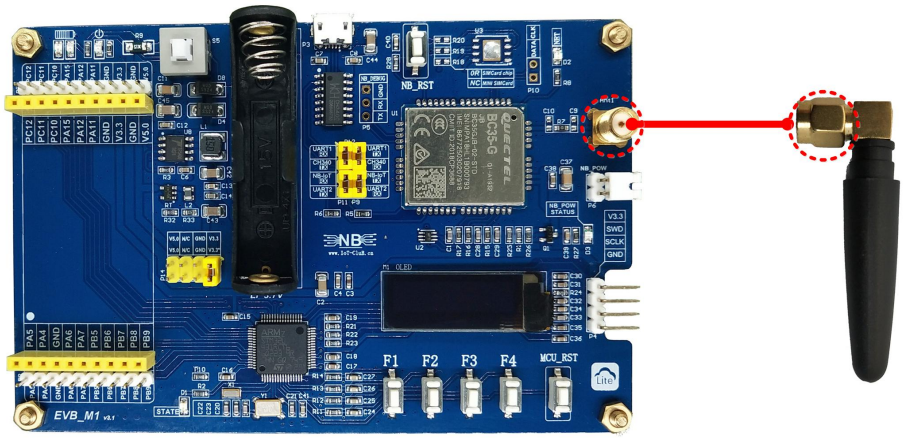
The EVB_M1 development board has the following resources:

- CPU: STM32L431RCT6, LQFP64; Flash: 128 KB, SRAM: 64 KB
- One power indicator (red)
- Two charging status indicators (red for charging and green for full power)
- One debugging indicator (green)
- One USB serial port, for the micro-control unit (MCU) log printing and NB-IoT module debugging
- One OLED for displaying debugging information
- Two reset keys for resetting the MCU and NB-IoT module respectively
- Four functional keys for man-machine interactions
- One power switch that controls the power supply of the entire board
- One extended port for connecting to different sensors
- One 10440 lithium battery holder with batteries

1.2 Installing the Experiment Platform of the Development Board

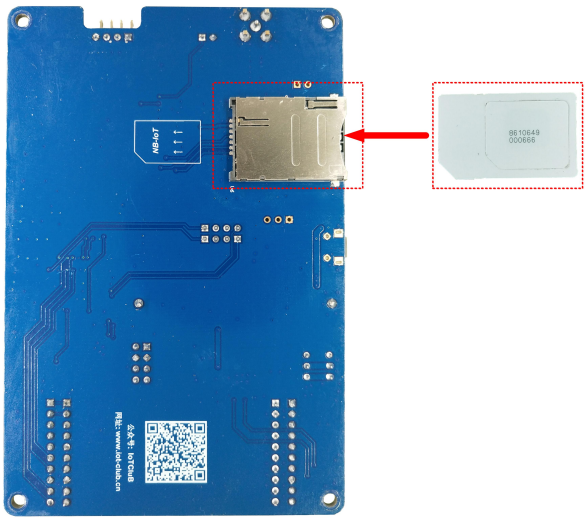
1.2.1 Installing an Antenna

Install the antenna on the development board.



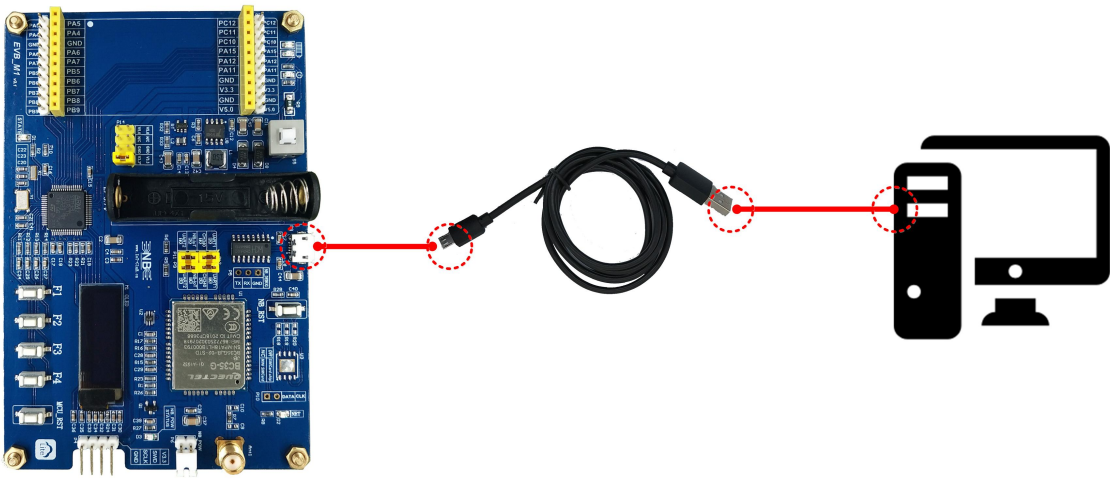
1.2.2 Installing an SIM Card

Prepare an SIM card, arrange it smoothly, and insert it to the development board. Note that the SIM card is inserted with its notch facing inward.



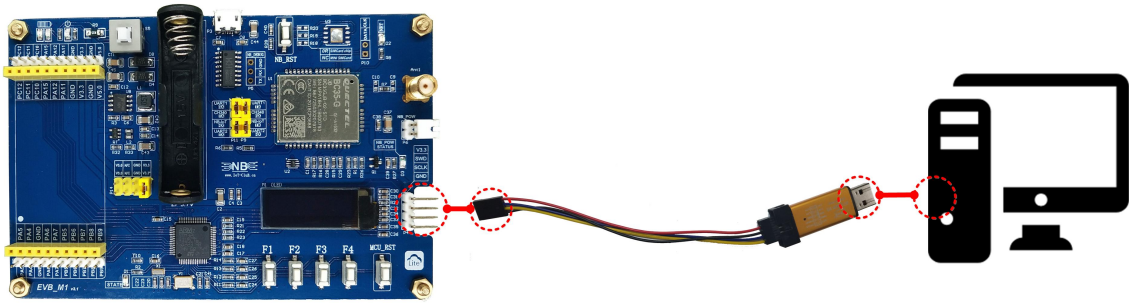
1.2.3 Connecting to a USB Cable

The micro USB cable is used for power supply and log printing. Connect one end of the cable to the micro port on the development board and the other end to the USB port of the computer.



1.2.4 Connecting to an ST-LINK Program Download Manager

Use 2 x 5-to-1 x 4 adapter cable for connection. End 1 x 4 is connects to the development board's program download interface, and the red color is VCC. End 2 x 5 connects to ST-Link program downloader



1.3 Introduction to the Hardware Circuits on the Development Board

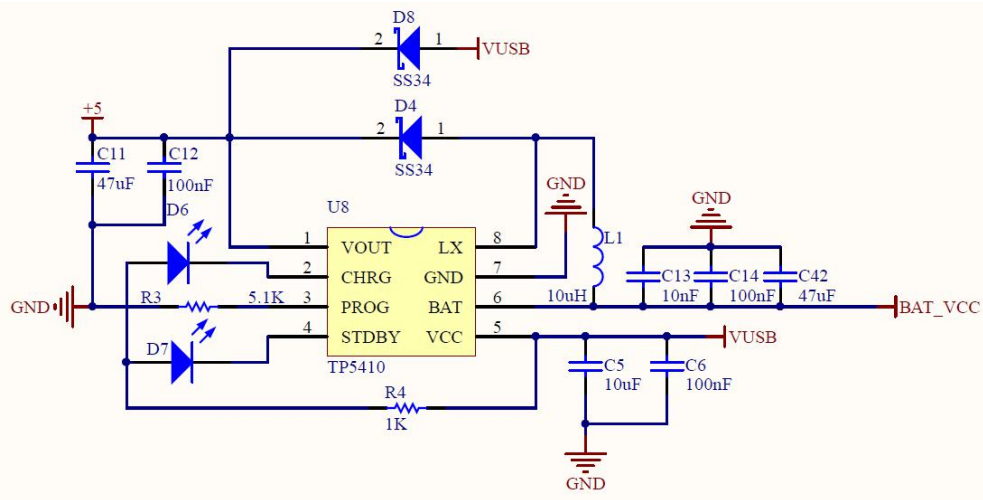
1.3.1 Power Circuits

The power circuit of the EVB_M1 development board consists of two parts: charge and discharge power management of the lithium battery and voltage regulator of the power supply.

1.3.1.1 Charge and Discharge Power Management of Lithium Batteries

To meet the requirements of project developers for later project applications, the lithium battery power supply module is added to the EVB_M1 development board. The lithium battery power supply module uses the TP5410 chip, which is a battery charging and 5 V constant voltage and voltage boost controller dedicated for a single lithium battery. The charging part integrates a high-precision voltage and charging current regulator, provides functions such as pre-charging, charging status indication, and charging completion indication, and outputs up to 1 A charging current. The voltage boost circuit uses the DC-DC boost converter of the complementary metal-oxide-semiconductor (CMOS) VFM switch type, which features the extremely no-load power consumption (less than 10 μA). The output drive current capability reaches 1 A.

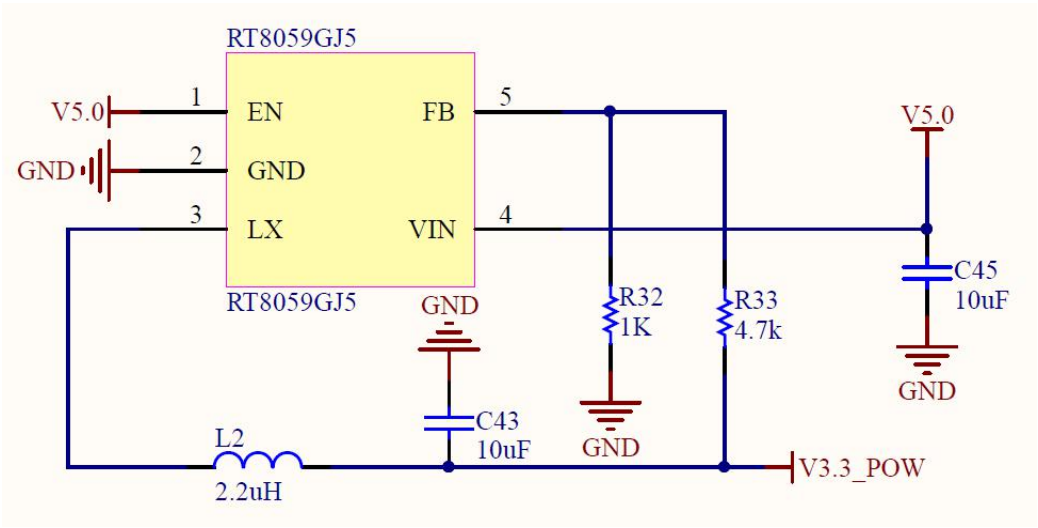
In the TP5410 circuit shown in the following figure, the VBUS provides voltage for the USB port, which is the total voltage. BAT_VCC is the charging voltage of the lithium battery. This pin is connected to the lithium battery. The VOUT pin outputs the +5 V voltage, which can be generated by the lithium battery and provided for subsequent circuits.



1.3.1.2 Voltage Regulator Circuit

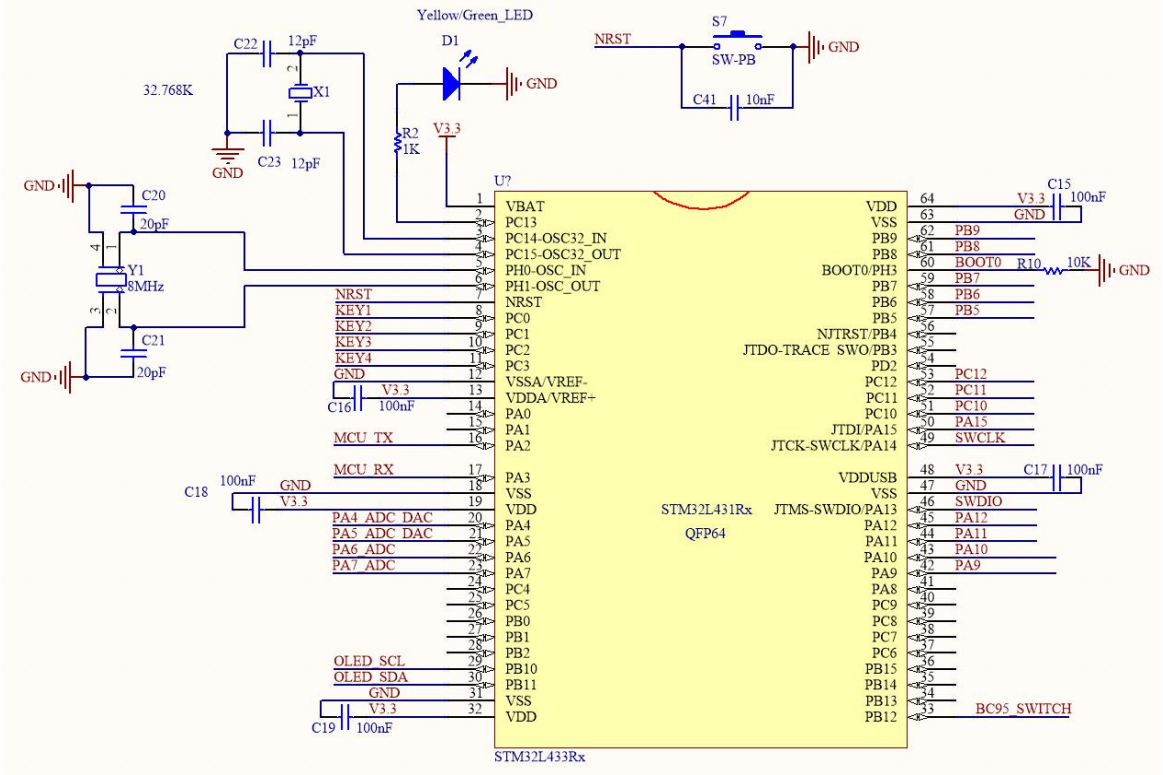
The power management circuit of the lithium battery can boost the 3.7 V lithium battery to 5 V, but the lithium battery after the boost cannot directly supply power to the EVB_M1. The power supply voltage range of components on the EVB_M1 is 1.8 V–3.6 V, and the recommended voltage is 3.3 V. (The power supply range of BC95 is 3.1 V–4.2 V, and the recommended voltage is 3.8 V. The 3.3 V power supply can also work properly.) Therefore, the 5 V level needs to be converted to 3.3 V for the EVB_M1 development board. RT8059 is used as the 3.3 V voltage regulator chip.

RT8059 is an efficient pulse width DC/DC buck converter. The input voltage is 2.8 V–5.5 V. The adjustable range of the output voltage is 0.6 V–Vin. The output current can reach 1 A. In the circuit provided by us, adjust the output voltage to 3.3 V for the development board to work normally.



1.3.2 Minimum System of the MCU

The minimum system of the MCU, also called the minimum hardware unit circuit, refers to the system that can work properly with the smallest number of components of the MCU. The minimum system consists of the power supply, MCU, crystal oscillator, reset circuit, and program burning interface. The 3.3 V power supply described in section 1.3.1.2is used. The following figure shows the schematic diagram of other parts.



1.3.2.1 Crystal Oscillator

STM32L431Rx can use either an internal or external crystal oscillator, which can be configured by software. The minimum system provides interfaces X1 and Y1 for connecting to the external 32.768 kHz crystal oscillator and external 8 MHz crystal oscillator, respectively.

1.3.2.2 Reset Circuit

The reset circuit is used to restore a circuit to the initial status. In the circuit diagram, the reset pin is connected to the GND through a key, and the key is connected to a capacitor in parallel to control the reset time. The reset time is relatively short and does not cause the abrupt voltage change on the pin.

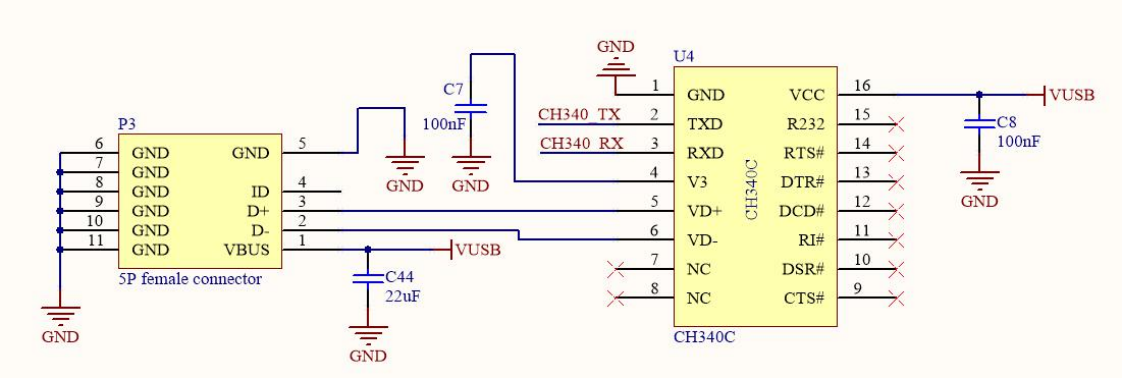
1.3.2.3 Program Burning Interface

An SWD four-wire interface is used as the program burning interface. The four wires are connected to the VCC, GND, SWDIO, and SWDCLK, respectively. An SWD interface is a serial debugging interface. Using the SWD interface not only improves reliability but also saves a lot of PCB space.

1.3.3 USB Level Conversion Circuit

The USB level conversion circuit is used to implement the communication between the MCU and PC. A USB interface is used as the communication interface on the PC, and the corresponding level logic has to comply with USB level rules. While a serial interface is used as the communication interface on the MCU, and the corresponding level has to comply with the TTL principle. To enable the MCU and PC to communicate with each other, a level converter is required. The EVB_M1 development board uses the CH340 chip as the converter. CH340 peripherals require only few components to transfer the USB bus.

It is widely used on the USB-to-TTL conversion tool due to ease of use. The following figure shows the circuit.



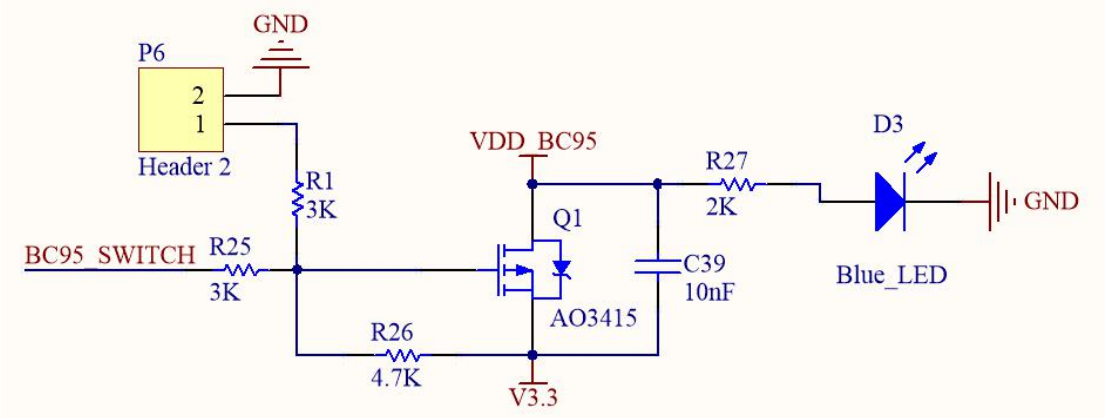
1.3.4 Peripheral Circuit of the NB-IoT Module

The peripheral circuit of the NB-IoT module consists of three parts: power supply circuit of the module, interface circuit, and SIM card circuit.

1.3.4.1 Module Power Supply Circuit

This is a power control circuit that enables the NB-IoT module to work. The circuit can be controlled by using either hardware or software. In hardware control, the power supply circuit is connected to P6 through the jumper cap. In software control, the BC95_SWITCH pin needs to be pulled down.

The power control circuit consists of one MOSFET. The gate of the MOSFET is pulled down so that power can flow through the MOSFET to the module. The circuit uses 3.3 V power generated by the power supply system described in section 1.3.1.2. At the power supply end, the circuit is connected to the GND through 100 nF and 47 uF capacitors. This connection mode supports clutter filtering while storing power, so that the module does not pull the power voltage below the working level when a large current is required.



1.3.4.2 Interface Circuit

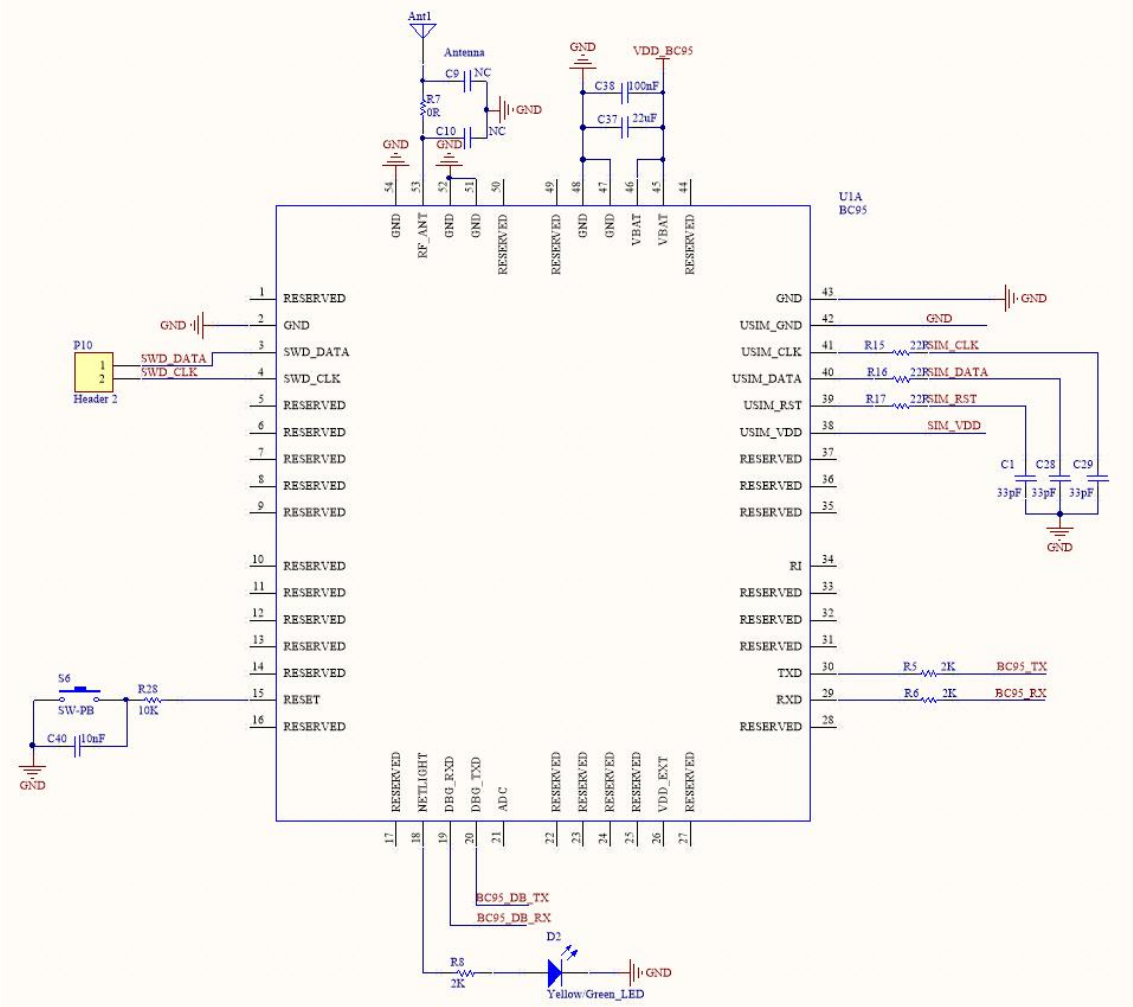
The interface circuit includes the antenna, reset circuit, communication interface, and other interfaces.

The antenna interface of the interface circuit is connected to the SMA antenna through the 0R circuit after it is led out of the module. Each end of the resistor is connected to a capacitor for grounding. The capacitor is not welded on the PCB board. Therefore, NULL is marked on the schematic diagram.

The reset circuit is used when parameters are configured to reset the module during debugging. The rest circuit provides the same functions as the reset circuit of the minimum system of the MCU in section 1.3.2.

The NB-IoT module has three types of interfaces: main serial interface, debugging serial interface, and SWD interface. The main serial interface is most frequently used for implementing communication between the MCU and module. After the main serial interface is led out from the EVB_M1 development board, it is connected to a row of pins. You can choose to connect it to the MCU or CH340 by using the jumper cap. The debugging serial interface is used to output logs. Generally, this interface is not used during development. To meet users' requirements, the debugging serial interface is led out separately from the EVB_M1 development board. The SWD interface is used for firmware burning. It is seldom used because the module currently only uses the main serial interface for firmware upgrades.

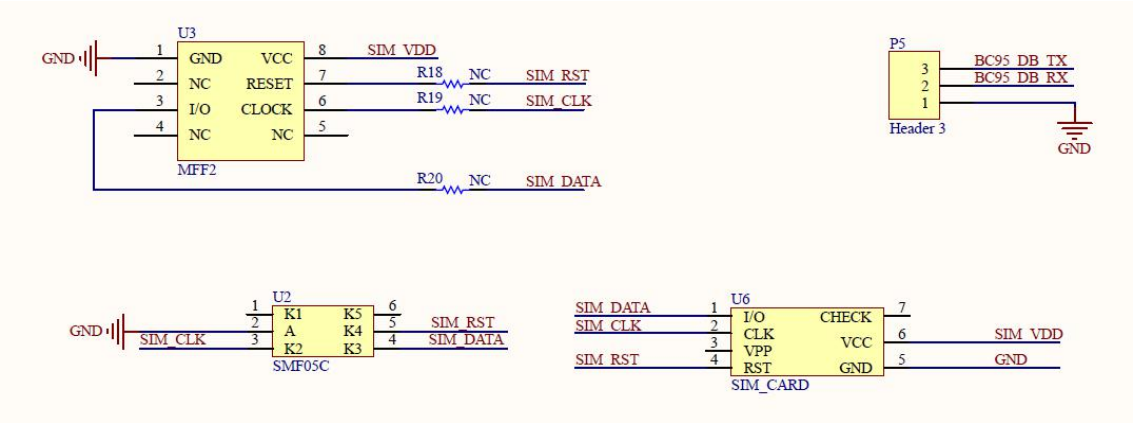
Other interfaces include ringing interfaces and network indicator interfaces. However, these interfaces are not implemented currently. They are only reserved on the development board.



1.3.4.3 SIM Card Circuit

Identity identification and authentication are required for the NB-IoT module to connect to the network properly. Same as mobile phone communication, the NB-IoT module also uses the cellular network, which requires an SIM card. An SIM card provides two packaging modes: common packaging (same as the SIM card used in mobile phones) and chip packaging. You can choose either of them.

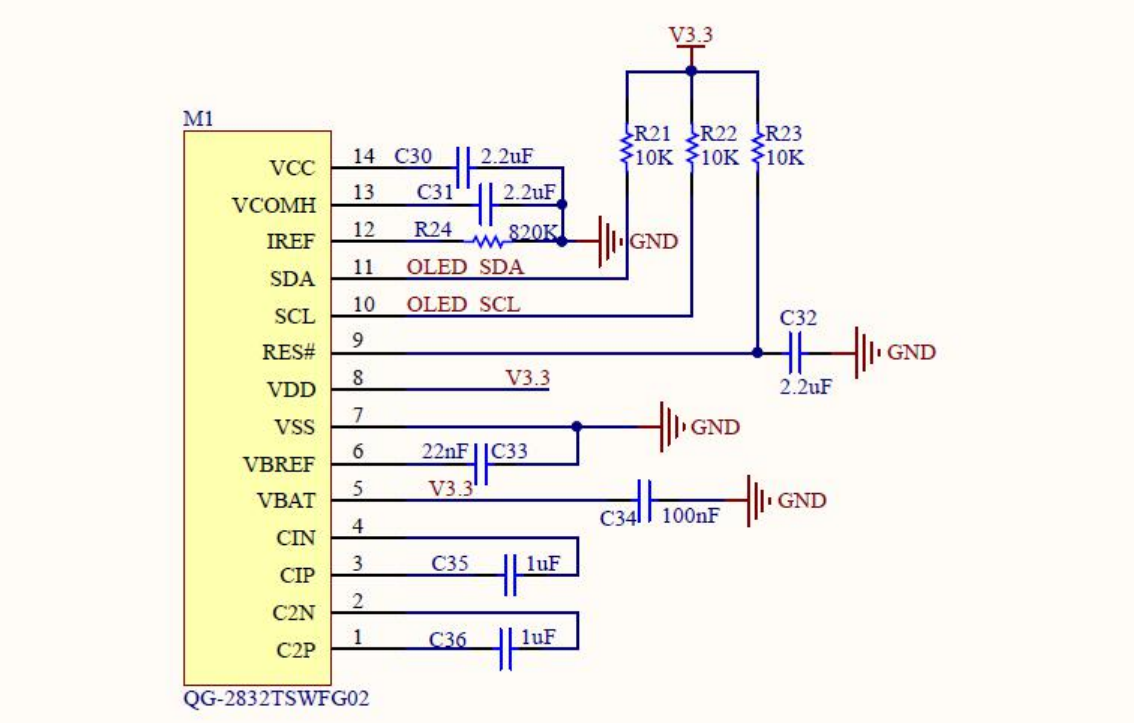
To switch an SIM card from common packaging to chip packaging, you need to connect R18, R19, and R20 on the M1 development board to the 0R resistor. Otherwise, disconnect the three resistors from the 0R resistor.



1.3.5 OLED

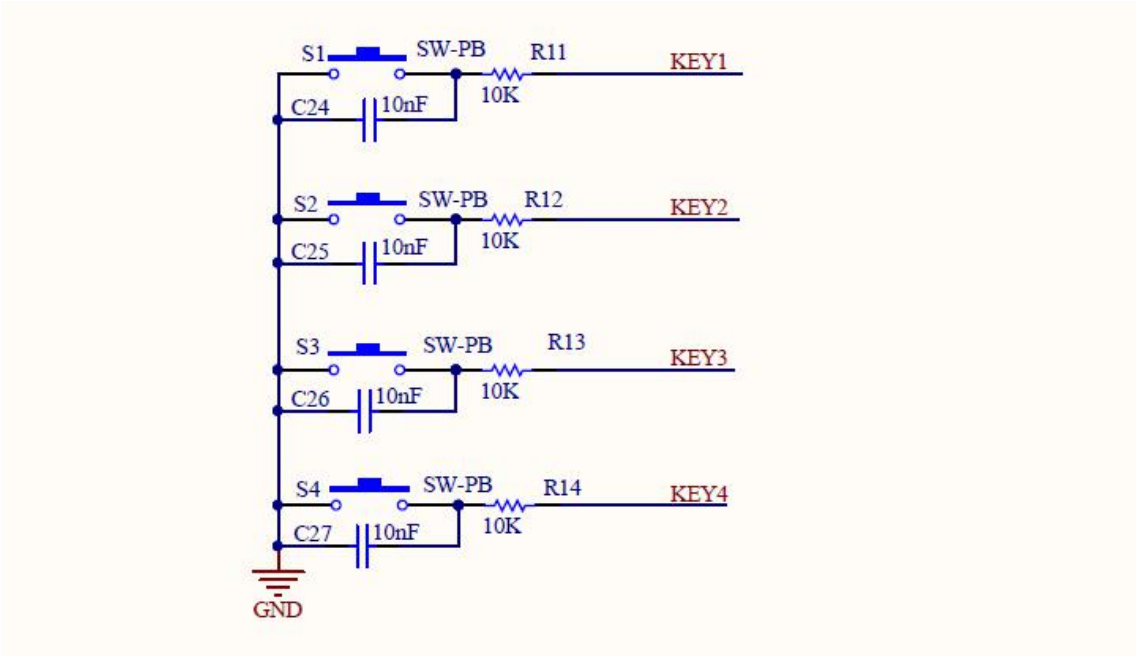
The OLED module displays the system status, parameters, or functions to be input to the system. To ensure good visual effects, the module uses the SSD1306-driven OLED with the resolution of 128 x 32. The SSD1306 chip is specially designed for the common-cathode OLED display. It embeds the comparison controller, display RAM, and crystal oscillator, reduces external components and power consumption, and support 256-level luminance control.

The OLED module uses the IIC interface. Because pins need to be pulled up when the IIC interface is idle, the module is connected to a 10 kilohm pull-up resistor and then to the MCU pin.



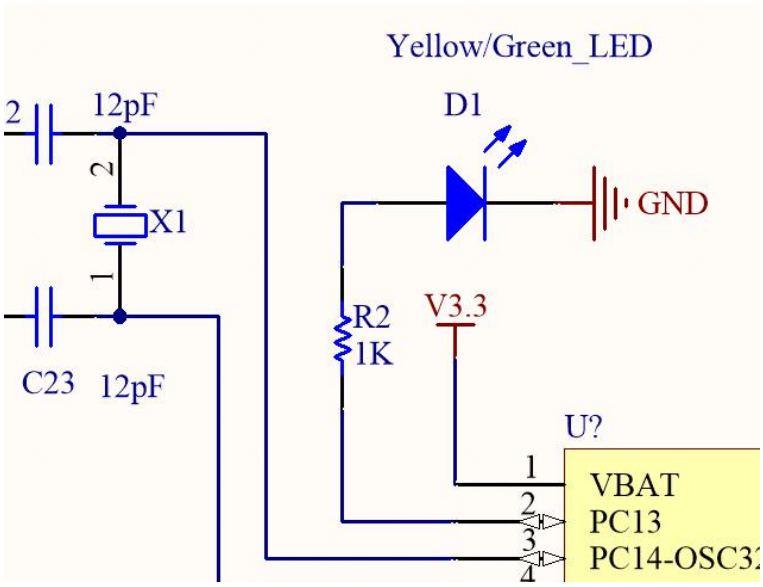
1.3.6 Keys

Keys are used as man-machine interfaces to input specific commands to the system. Because the number of keys is small, the EVB_M1 development board uses independent keys to access the system. In this way, the system responds quickly. In addition, the functions of keys can also be implemented by using interrupts. Users themselves can control engineering by compiling key functions.



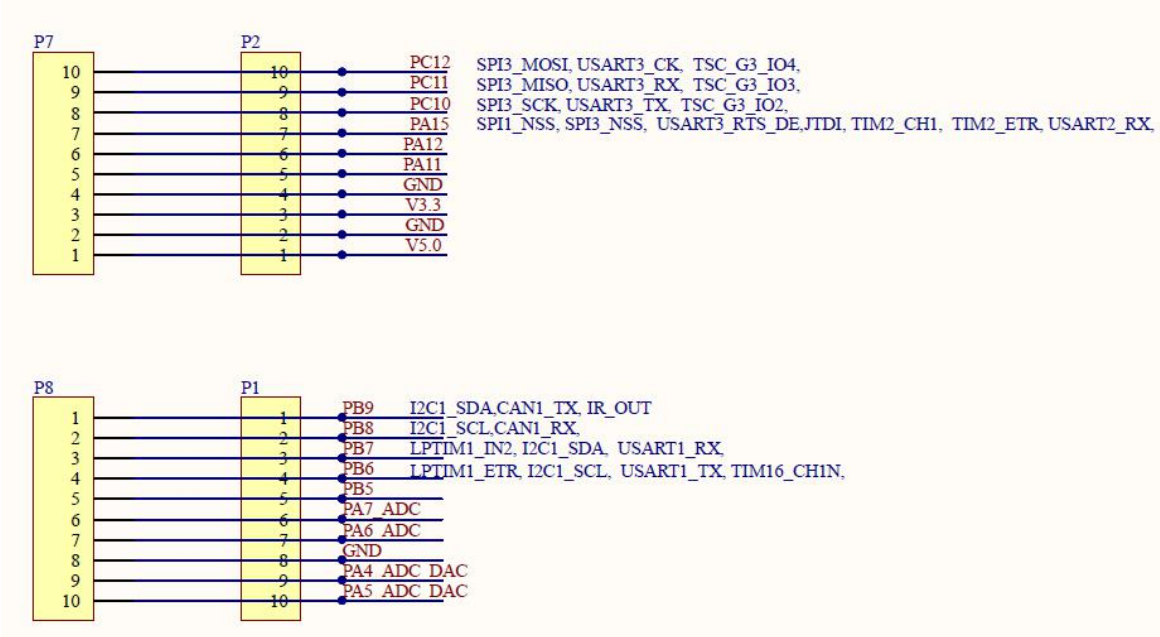
1.3.7 LED Indicator

To facilitate project development and debugging, the EVB_M1 development board connects to an LED for status setting. The LED indicator is connected to the PC13 pin of the MCU. When the PC13 pin outputs high level, the LED indicator will be on.



1.3.8 Extended Interfaces

Extended interfaces are reserved on the EVB_M1 development board to meet the expansion requirements of developers. These interfaces can be connected to the extension board of EVB_M1. Alternatively, they connect to their sensors by using functions of the corresponding pins of the MCU. The following schematic diagram shows the general functions of these pins. If you want to use their more functions, use them in combination with STM32CubeMX.



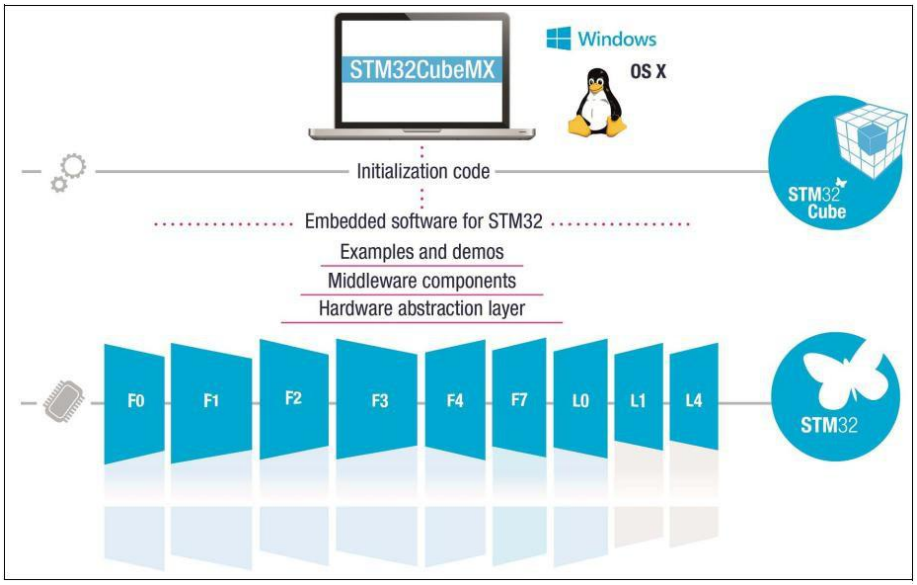
The preceding sections describe the hardware of the experiment platform (EVB_M1 development board). After learning the whole hardware, you will understand the code introduced later better and find it easier to compile software. To obtain other documents and tutorials of the EVB_M1 development board, visit the technical forum at www.iot-club.cn.

2 Software

This chapter describes how to install the software and drivers required by the development board, how to set up the development environment, and how to obtain the Huawei IoT platform.

2.1 STM32CubeMX Installation and Development Environment Configuration

2.1.1 Introduction to STM32CubeMX



STM32CubeMX (STM32Cube initialization code generator) is a graphical tool that automatically generates the corresponding initialization code related to chips at the initial stage of development.

As shown in the preceding figure, it contains all series chips of the STM32 microcontroller, including examples and demos, middleware components, and the hardware abstraction layer.

STM32CubeMX has the following features:

1. Intuitive GUI for selecting the STM32 microcontroller
2. Graphical configuration of the microcontroller
   - Automatically processes pin conflicts.
   - Dynamically sets the determined clock tree.
   - Dynamically determines parameters for setting the peripheral and middleware modes and initialization.
   - Predicts power consumption.
3. Coverage of the initialization compilation software of the STM32 microcontroller, such as IAR, KEIL, and GCC
4. Used separately or as an Eclipse plug-in

STM32CubeMX is an active original tool developed by STMicroelectronics, which makes developers' lives easier by reducing development effort, time and cost. STM32CubeMX embeds a comprehensive software platform that supports the development of each series MCU of STM32. This platform includes STM32Cube HAL (an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio), plus a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics. All the embedded software utilities are delivered with a full set of examples.

For more information about STM32CubeMX, visit the STMicroelectronics official website at https://www.st.com/.

2.1.2 Installing the JRE

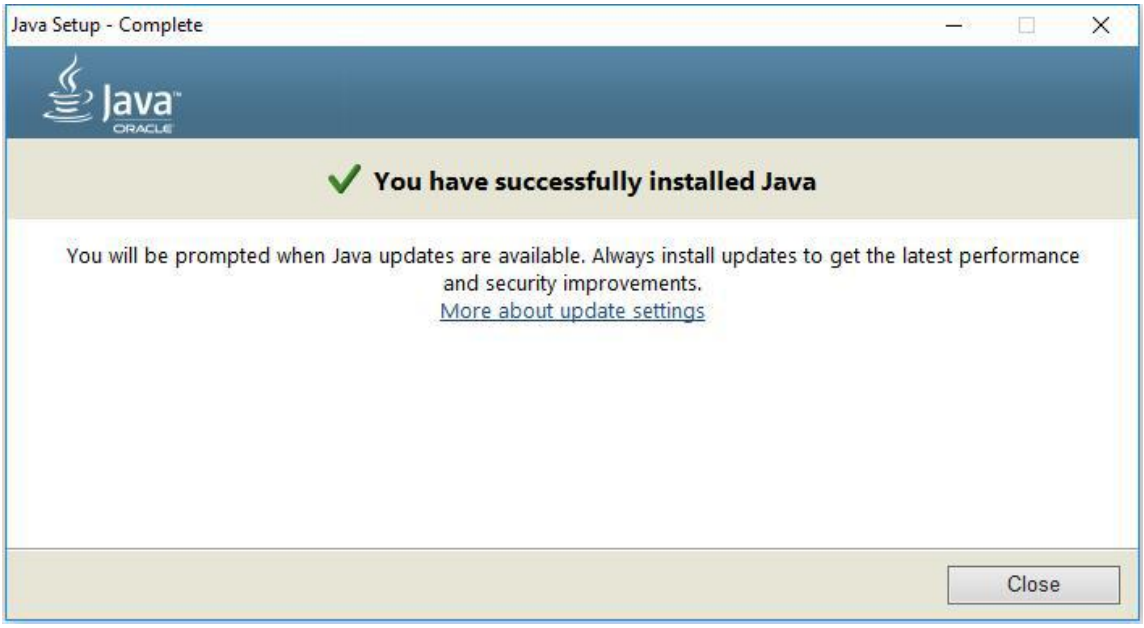
STM32CubeMX is running on the Java environment. Therefore, you need to install the JRE first. The latest JRE version is jre1.8.0_131. STM32CubeMX requires that the minimum JRE version is 1.7.0_45. If the version of the JRE installed on your computer is later than 1.7.0_45, you do not need to download and install the latest JRE.

The JRE is easy to install. The latest version **jre-8u112-windows-i586.exe** is used as an example to describe the installation process.

Step 1 Double-click **jre-8u112-windows-i586.exe**. (If the administrator permission is required, right-click **jre-8u112-windows-i586.exe** and choose **Run as Administrator**.) Click **Install**. If you do not change the target folder, the JRE is installed in **C:\Program Files\Java\jre1.8.0_112** by default.

Step 2 Wait for the installation to complete.

Step 3 Click **Close** after the installation is complete.



----End

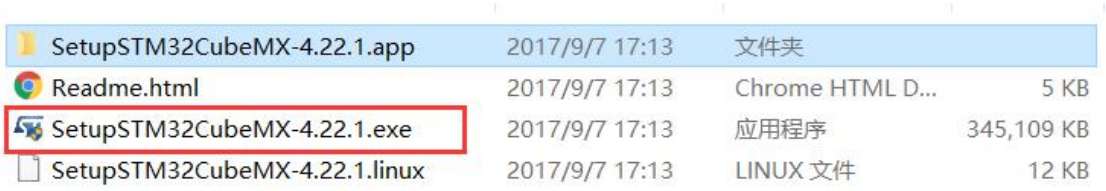
2.1.3 Installing STM32CubeMX

The STM32CubeMX version provided by us is V4.22.1, which supports the following OSs:

- Windows XP: 32-bit (x86)
- Windows 7: 32-bit (x86), 64-bit (x64)
- Windows 8: 32-bit (x86), 64-bit (x64)
- Linux: 64-bit (x64) (tested on RedHat, Ubuntu and Fedora)
- MAC OS X Yosemite or later

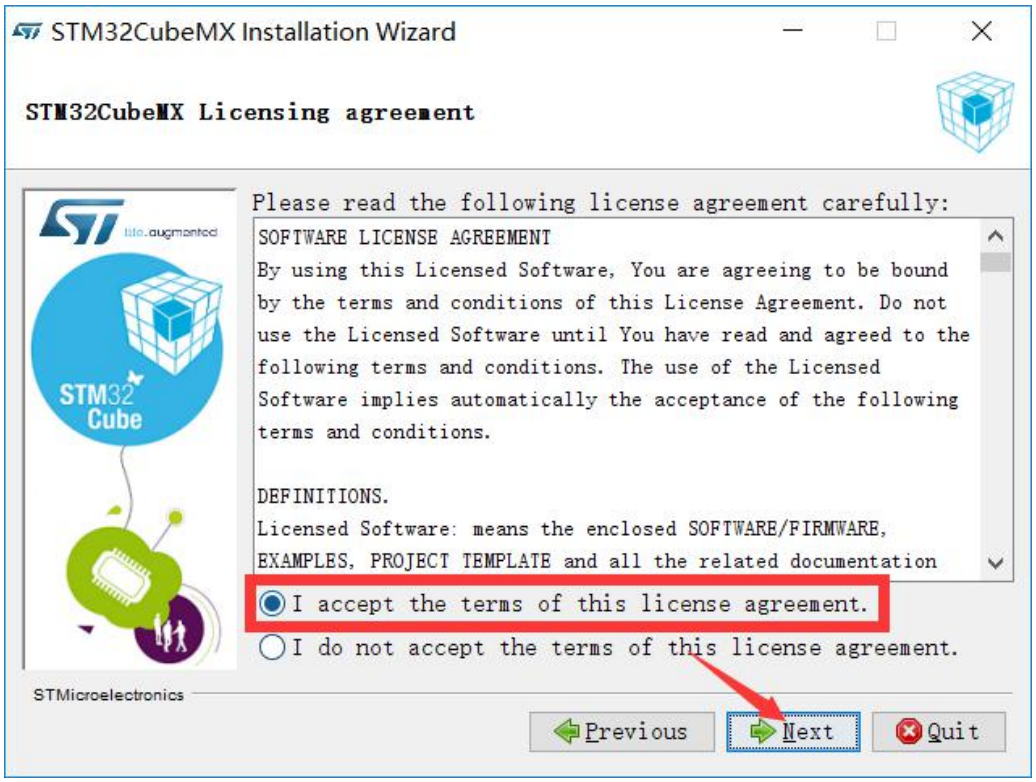
The following describes how to install STM32CubeMX easily:

Step 1 Decompress the downloaded file and double-click **SetupSTM32CubeMX-4.22.1.exe**. If the administrator permission is required, right-click **SetupSTM32CubeMX-4.22.1.exe** and choose **Run as Administrator.**

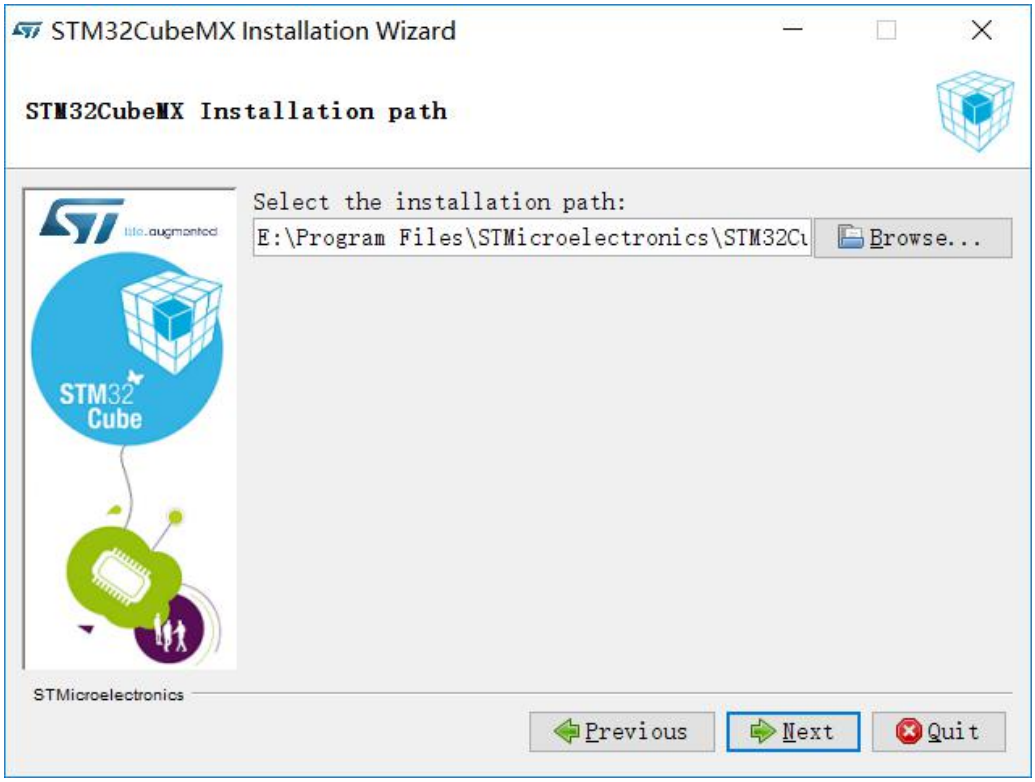


Step 2 Click **Next**.

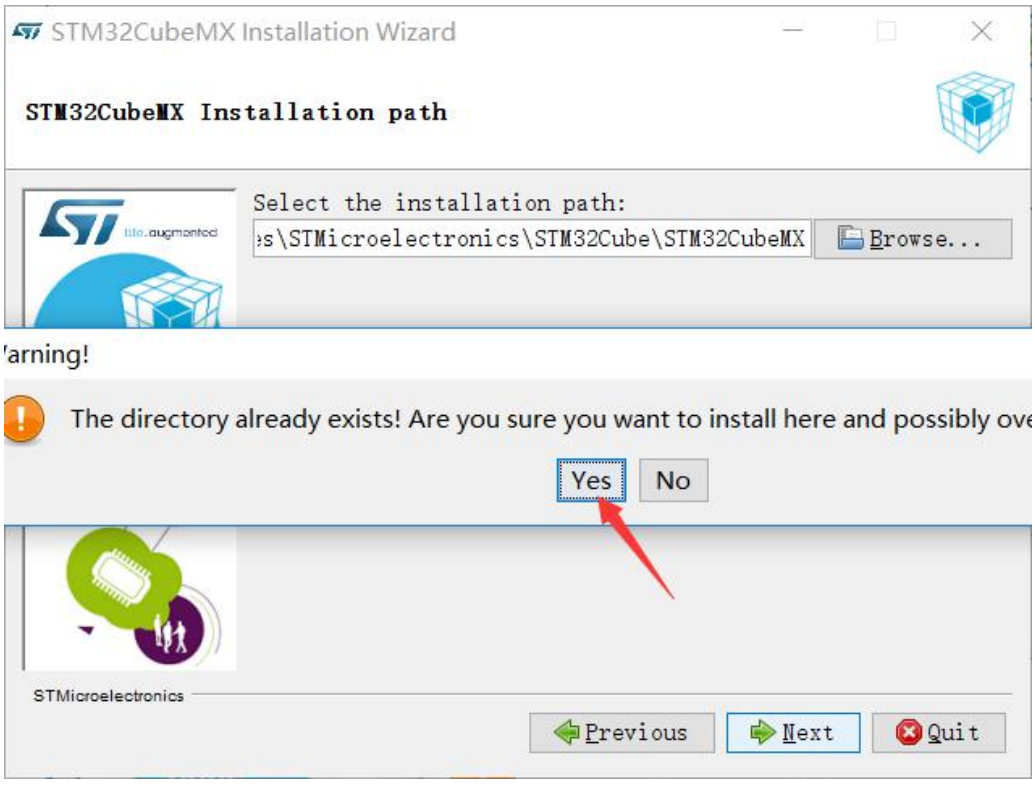
Step 3 Select **I accept the terms...** and click **Next**.



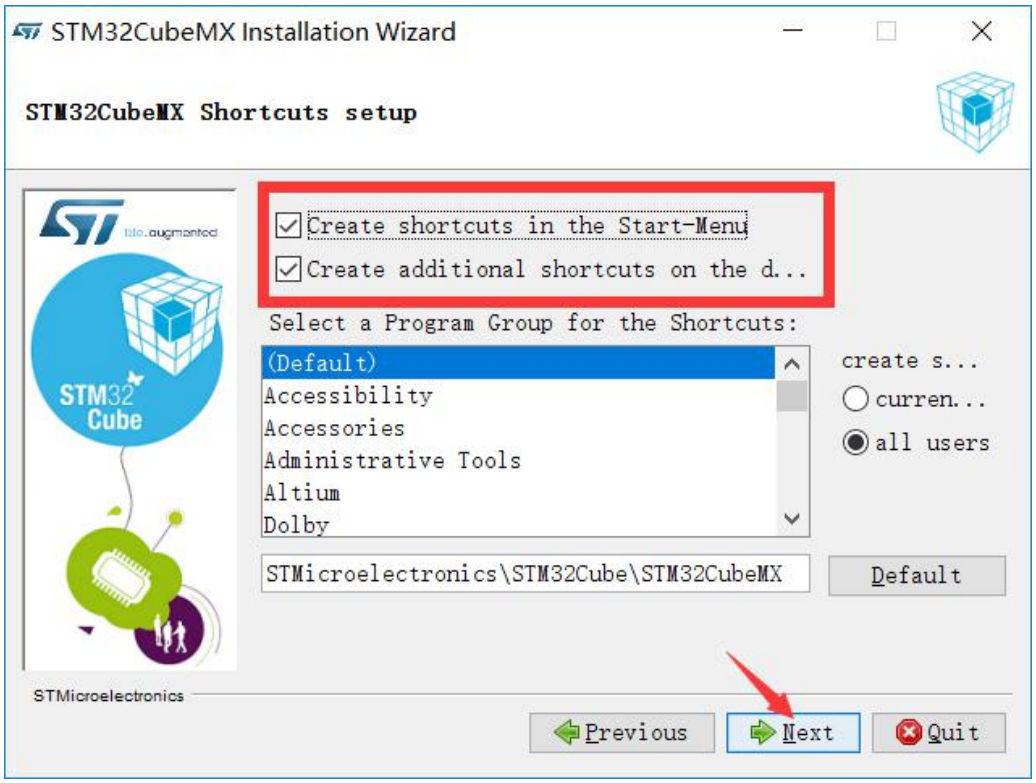
Step 4 Select an installation path and click **Next**. (The following uses a path on drive E as an example.)



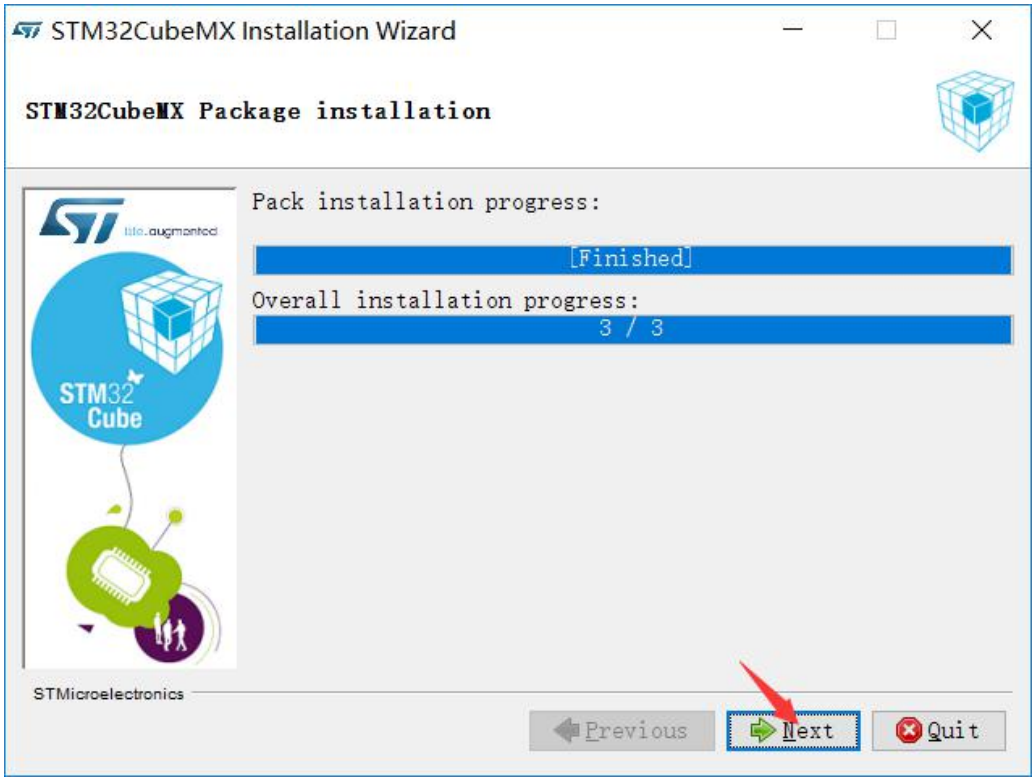
Step 5 In the dialog box that is displayed, click **Yes**.



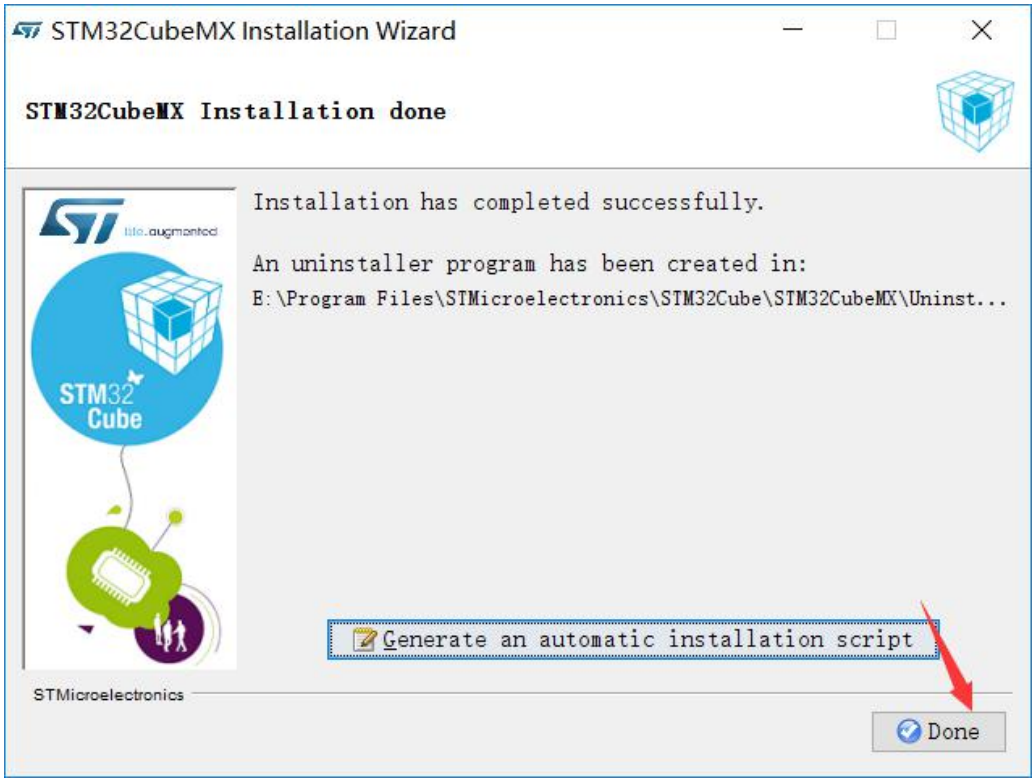
Step 6 Select **Create shortcuts in the Start-Menu** and **Create additional shortcuts on the d…** and click **Next**.



Step 7 After the installation is complete, click **Next**.



Step 8 Click **Done**. The STM32CubeMX installation is complete.



----End

The preceding operation procedure shows the installation of STM32CubeMX on a Windows OS. For details about how to install it on the Linux or Mac OS, see the description provided on the official website at https://www.st.com/.

On Linux:

Make sure you have access rights to the target installation directory, you can run the installation as root (or sudo) to install STM32CubeMX on shared directories Double-click (or launch from the console window) on the SetupSTM32CubeMX-4.17.0.linux file

On Mac OS:

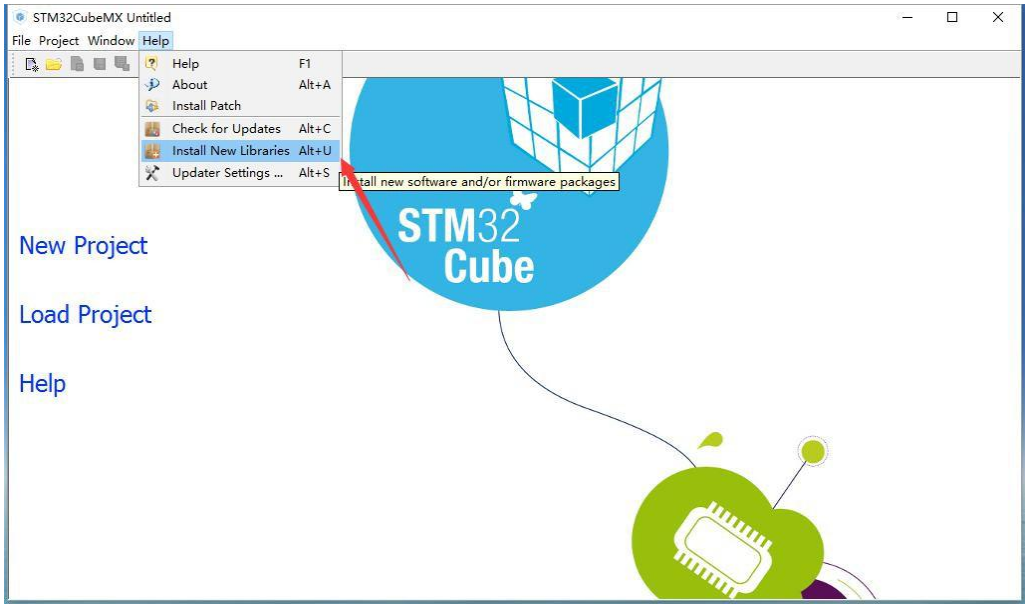
Make sure you have access rights to the target installation directory, you can run the installation as root (or sudo) to install STM32CubeMX on shared directories Double-click (or launch from the console window) on the SetupSTM32CubeMX-4.17.0 application

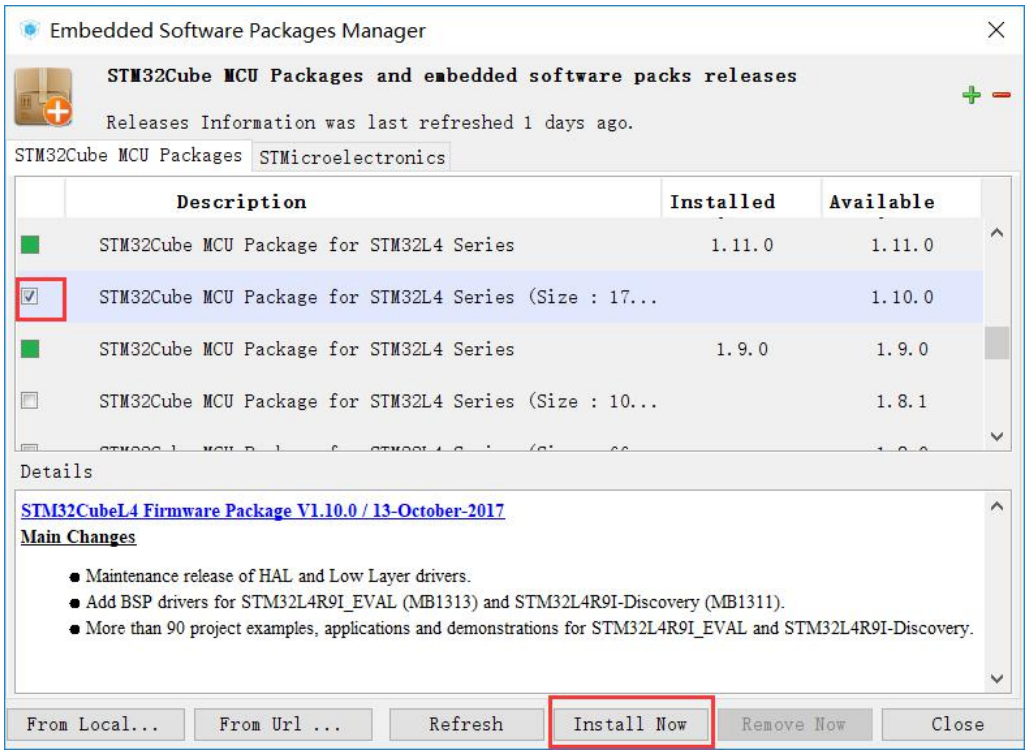
2.1.4 Installing the STM32CubeMX Library

You can install the STM32CubeMX library (firmware support package) using STM32CubeMX in the following three modes: online installation, import of the offline package, and decompression of the offline package.

- Online installation

  Open the installed STM32CubeMX. Choose **Help** > **Install New Libraries** to access the library management page. In the list that is displayed, select the firmware library to be installed and click **Install Now** until the installation is successful.





The description of STM32CubeMX installation and environment configuration is complete. STM32CubeMX is a mandatory tool for developing STM32. You are advised to learn STM32CubeMX, because it may be the mainstream tool for developing STM32 in the future.

2.2 SW4STM32 Software Installation and Development Environment Configuration

2.2.1 Introduction to SW4STM32

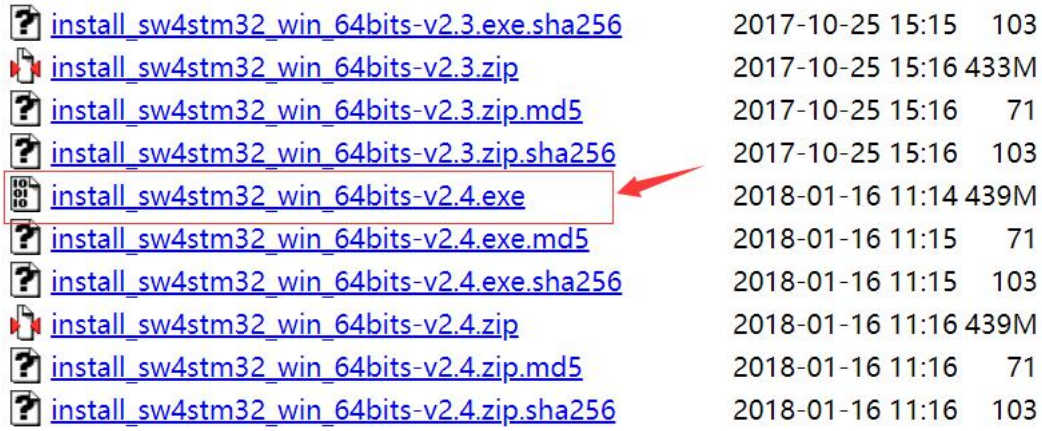
SW4STM32 is an integrated development environment (IDE) developed by AC6 based on the Eclipse graphical interface. SW4STM32 uses the GCC-based compilation and debugging toolchain, which is

free of charge and has no limit on the size. You can configure the source code path and GCC compilation options on the graphical interface of SW4STM32. This helps you quickly jump to other code from the source code, supplement the code during compilation, and view parameters. SW4STM32 supports the import of project files generated by STM32CubeMX. The project files can be directly compiled without modification. SW4STM32 supports the ST-Link with the OpenOCD. You can debug the source code step by step based on the graphical interface.

2.2.2 Installing SW4STM32

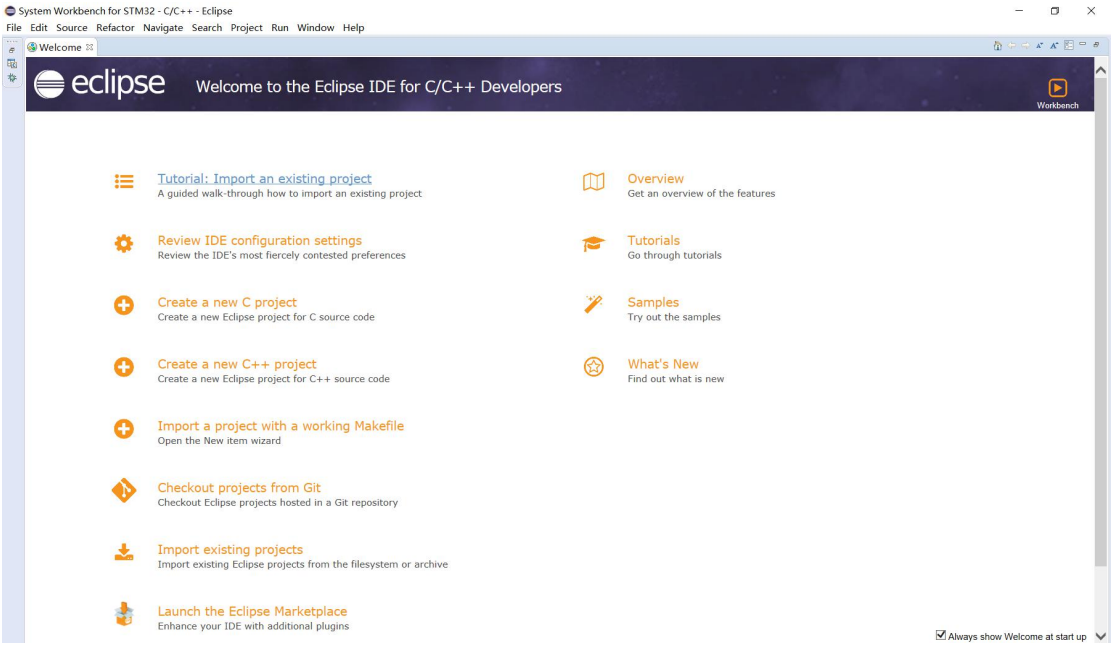
2.2.2.1 Downloading SW4STM32

You can download the latest windows platform installation package of SW4STM32 from http://www.ac6-tools.com/downloads/SW4STM32/, as shown in the following figure.



2.2.2.2 Installing SW4STM32

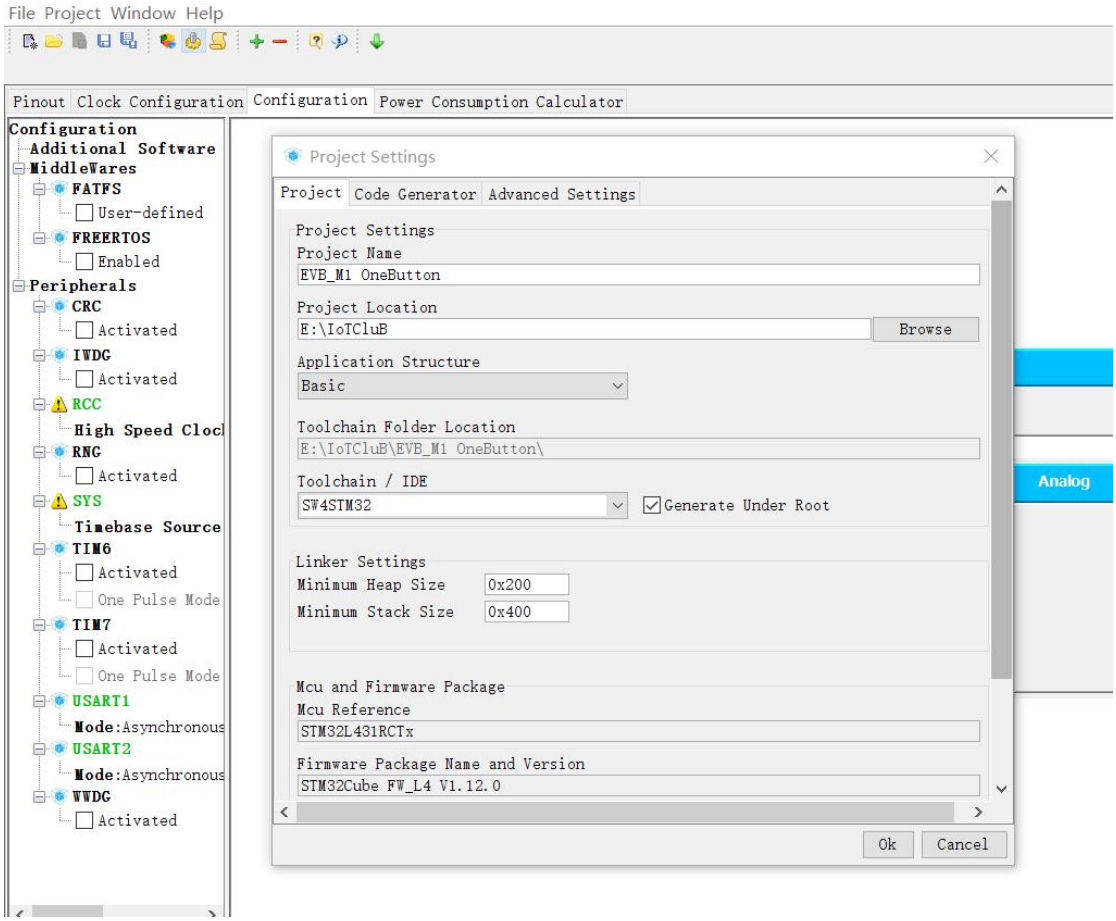
The installation is simple. You can double-click the installation package to install SW4STM32. The Java runtime environment (JRE) is required. After the installation is complete, the Eclipse window shown in the following figure is displayed after you close the welcome page.



2.2.2.3 Creating a Project

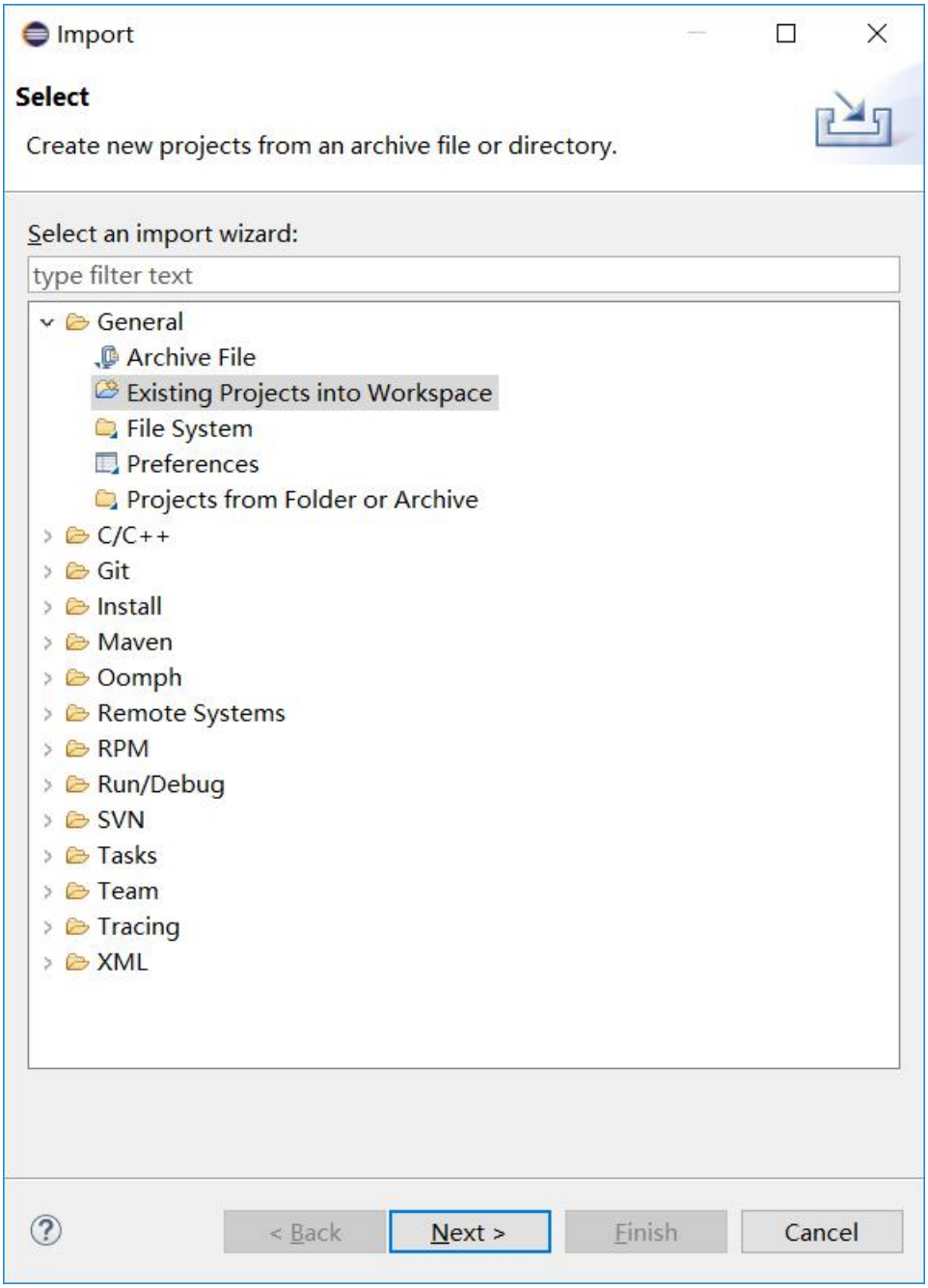
2.2.2.4 Generating STM32CubeMX

The recommended software STM32CubeMX is used to create a project. Currently, the latest STM32CubeMX supports direct generation of an SW4STM32 project. The following figure shows the configuration.

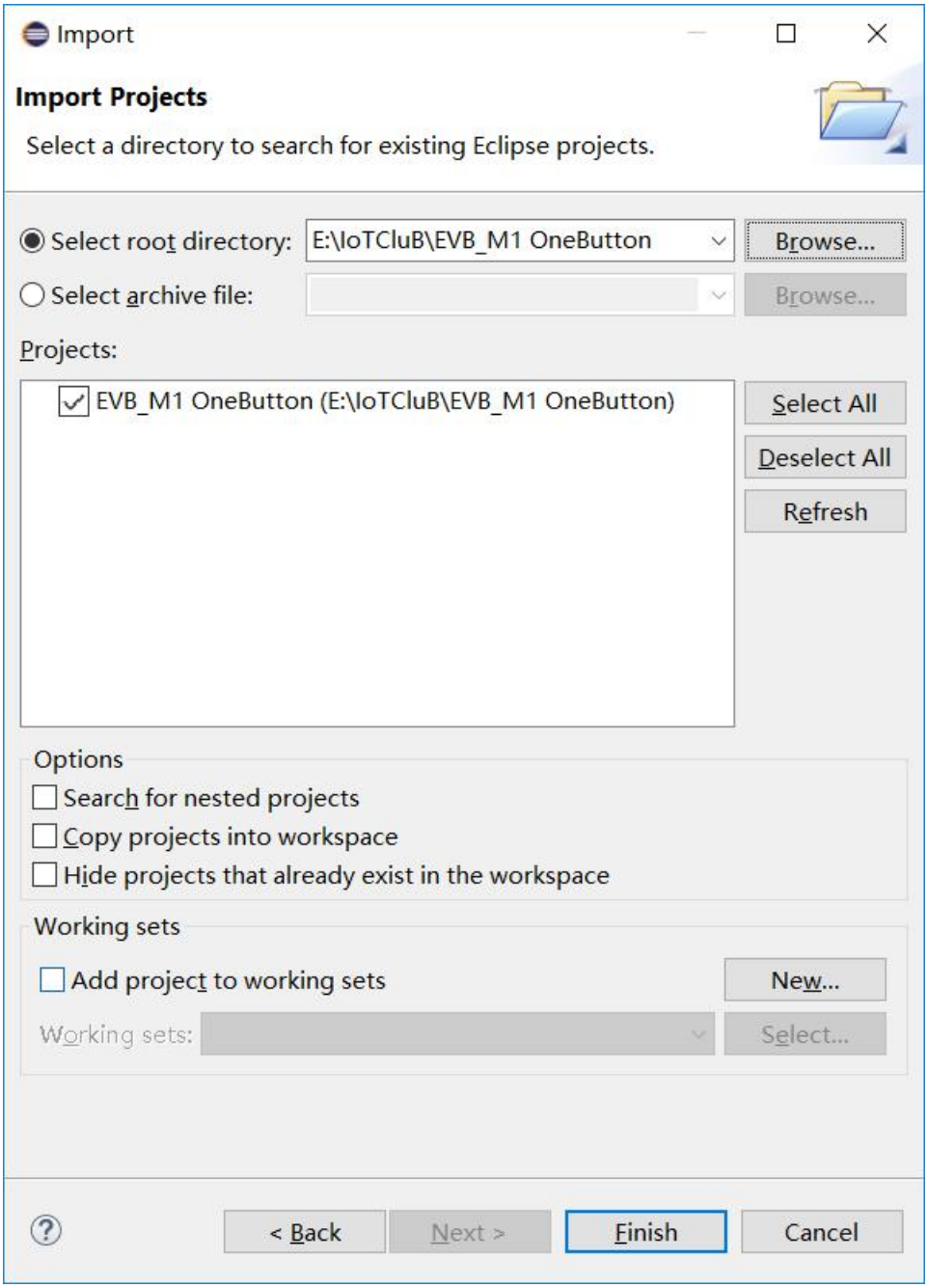


2.2.2.5 Importing a Project

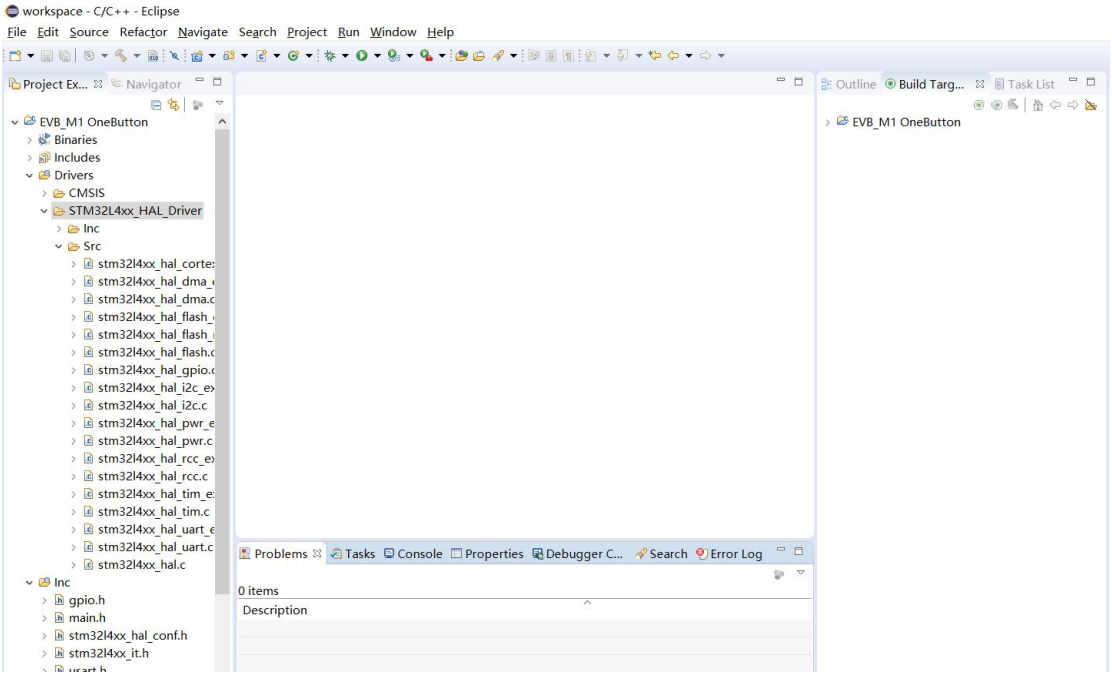
Open SW4STM32 and choose **File** > **Import**. In the displayed dialog box shown in the following figure, click **Next**.



Specify the directory for storing project information.



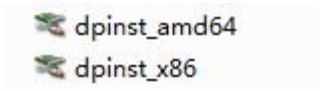
Click **Finish**. The project has been imported, as shown in the following figure.



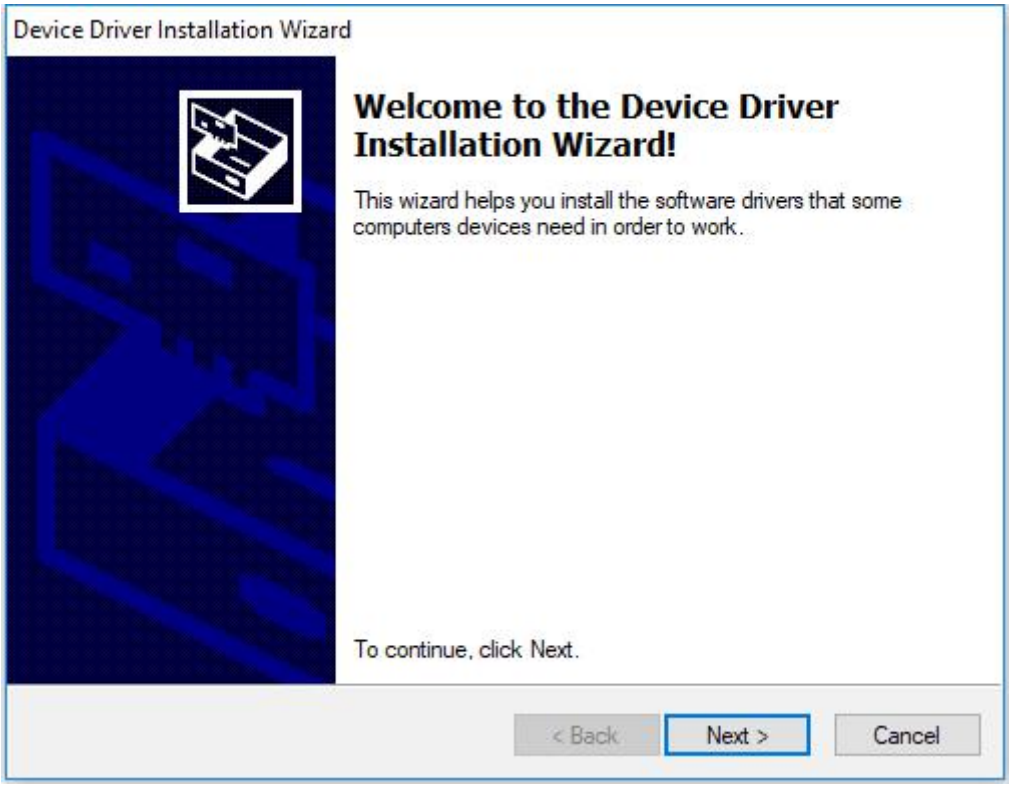
2.2.3 Installing the ST-LINK Driver

The setup of the development environment for the development board's MCU program has been described in Chapter 1. However, to burn the program to the development board, an emulator is required. The development board uses the ST-LINK/V2 emulator provided by STMicroelectronics to burn and emulate the development board program. The following describes how to install the ST-LINK driver and set up the environment.

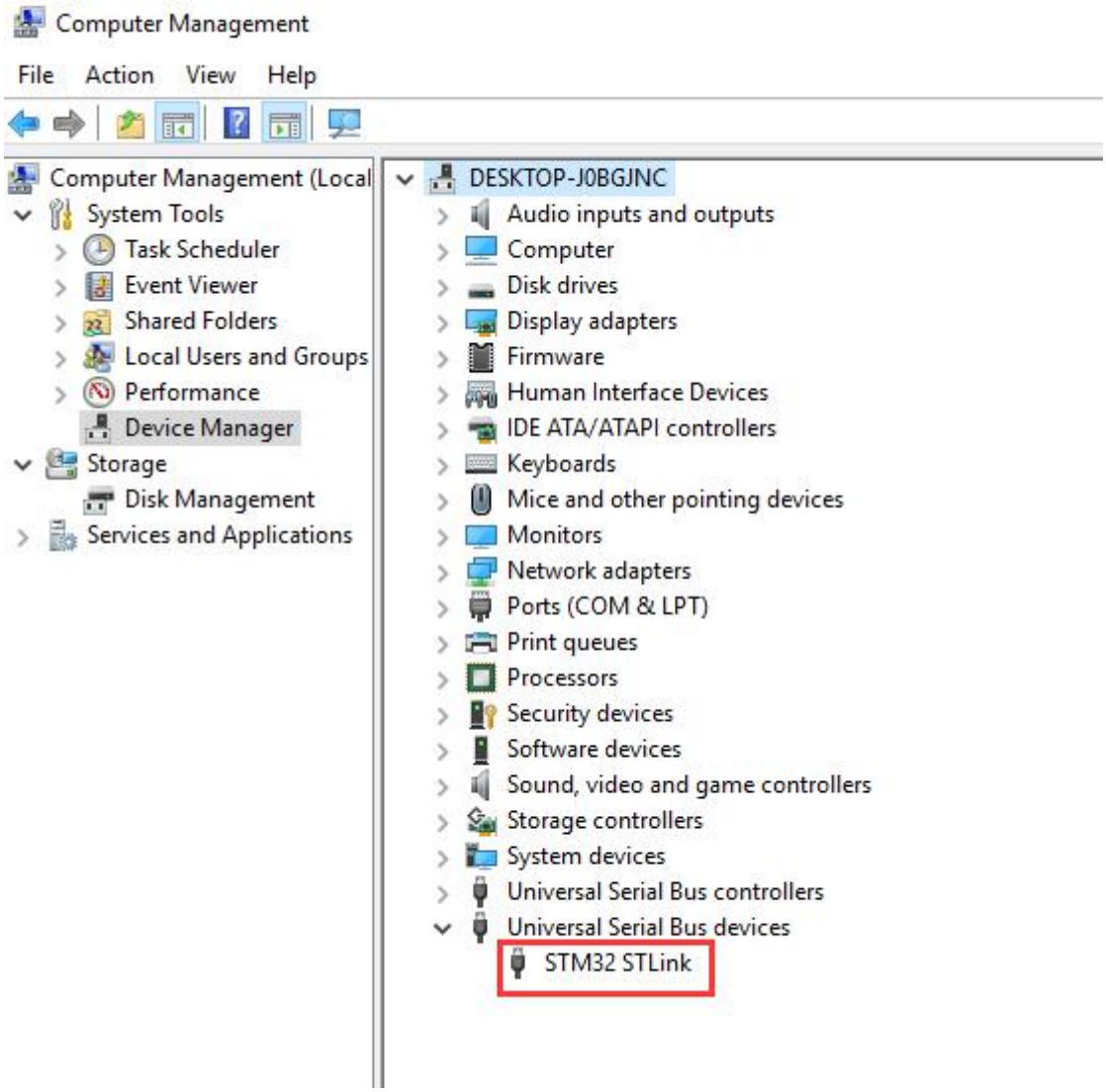
- In **EVB_M1\05 Driver\STLINK V2,** there are two types of drivers.



  For a 32-bit computer, install dpinst_x86. For a 64-bit computer, install dpinst_amd64.
- Install an ST-LINK/V2 driver as required. You are advised to use the default installation path.



- After the installation is complete, connect the ST-LINK to the computer through the USB port. Open the **Device Manager** window. If **STM32 STLink** is displayed, the driver is installed successfully.



NOTE

- The device name and the node in the ***Device Manager*** *window* may vary according to Windows versions. For example, if the ST-LINK is installed on Windows 10, **STM32 STLINK** is displayed.
- If a yellow exclamation mark (!) is displayed next to the device name, click the device name. In the displayed dialog box, click to update the device driver.

After the preceding operations are performed, the ST-LINK/V2 driver has been installed.

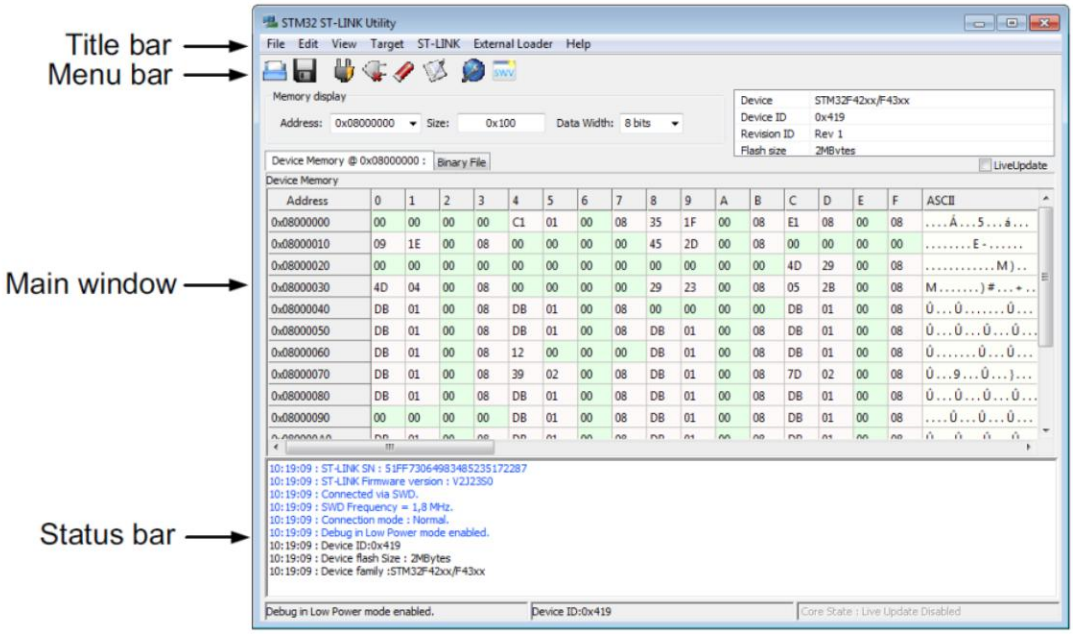
2.2.4 Installing the STM32 ST-LINK Utility

2.2.4.1 Introduction to the STM32 ST-LINK Utility

The STM32 ST-LINK Utility supports the download of code in batches. When you need to view flash data of a chip, you can quickly locate the desired data with the **STM32 ST-LINK Utility.** (The prerequisite is that no protection is added.)

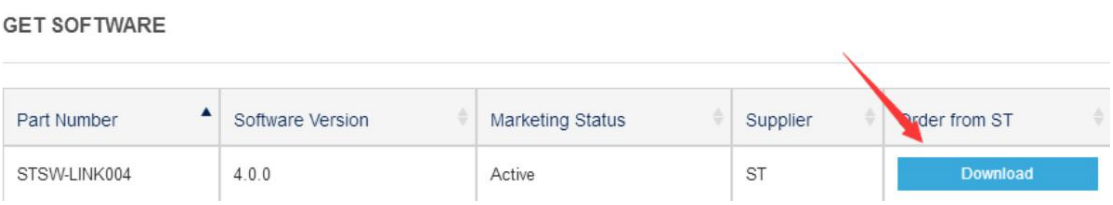
The STM32 ST-LINK Utility contains the ST-LINK/V2 driver. If you have installed the STM32 ST-LINK Utility, you do not need to install the driver for the ST-LINK. You can directly use it functions (such as Keil, IAR online debugging, and download).

Besides flash data, the STM32 ST-LINK Utility can quickly read the model, ID, and version of the STM32 chip.



2.2.4.2 Downloading the STM32 ST-LINK Utility

Up to November, 2018, the latest version of STM32 ST-LINK Utility is V4.0.0. To download it from the official website at http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link004.html, an ST account must be registered.



2.2.4.3 Installing the STM32 ST-LINK Utility

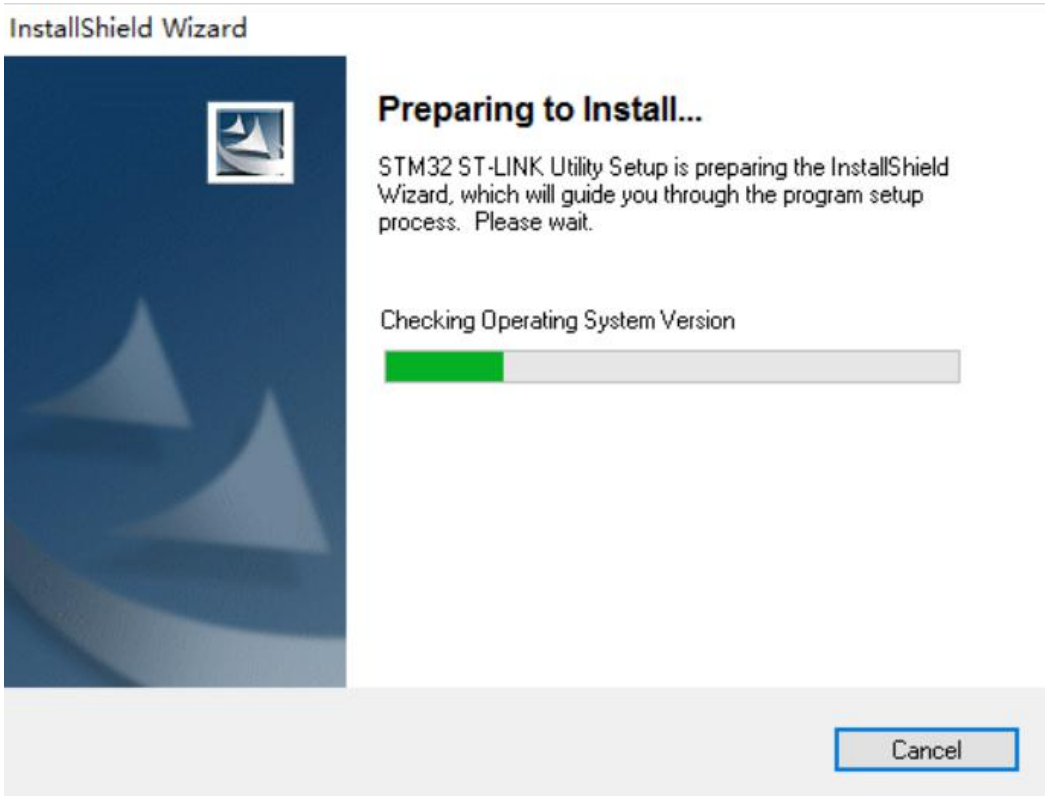
The IDE the STM32 ST-LINK Utility is easy to install. For details, see the common installation tutorial presented using the following screenshots. The downloaded **STM32 ST-LINK Utility v4.0.0 setup.exe** is used as an example.

The environment requirements are as follows:

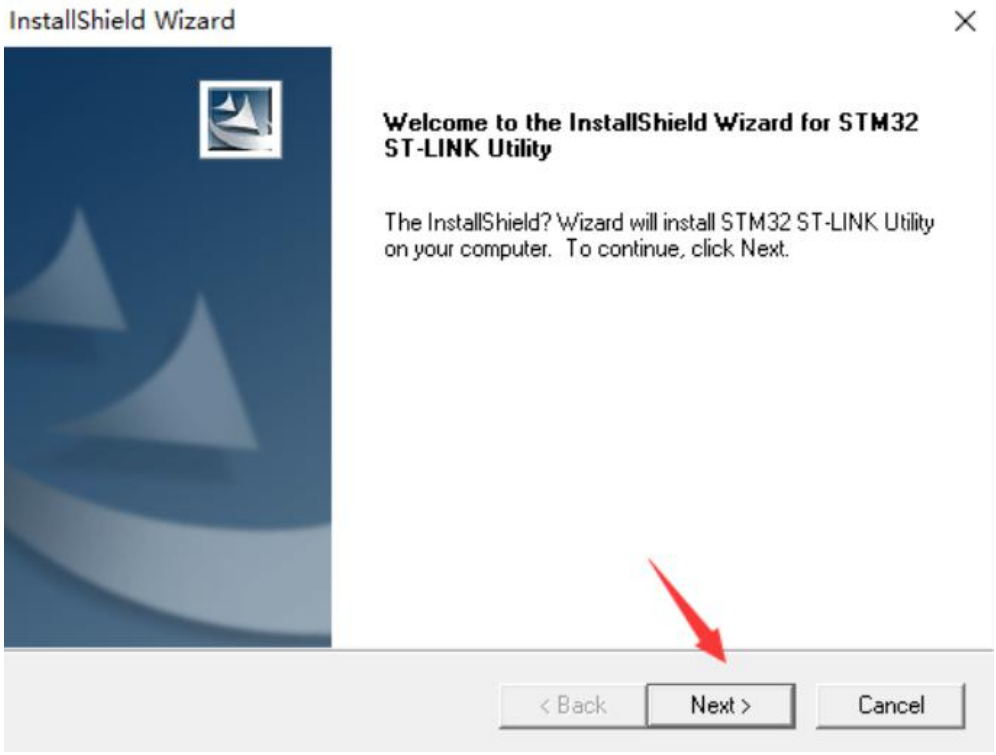
- OS: Windows XP, Windows 7, and Windows 10
- Memory space: 256 MB or above
- Disk space: 30 MB or above

Perform the following steps to install the STM32 ST-LINK Utility:

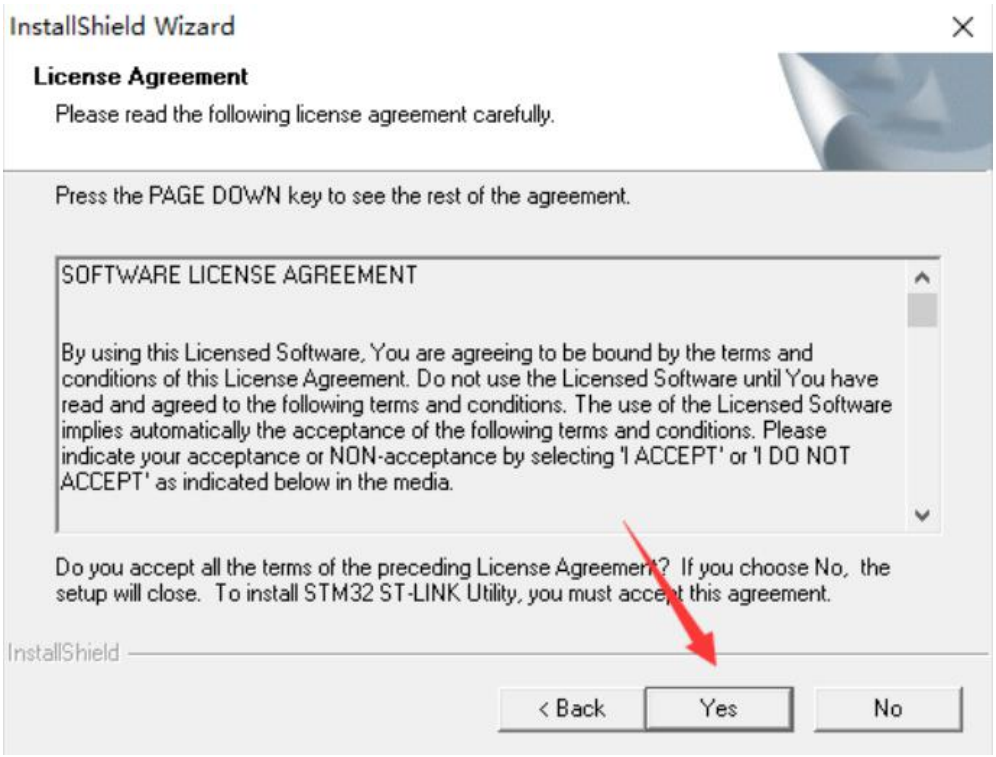
Step 1 Decompress the software package and double-click **STM32 ST-LINK Utility v4.0.0.exe** to start the installation.



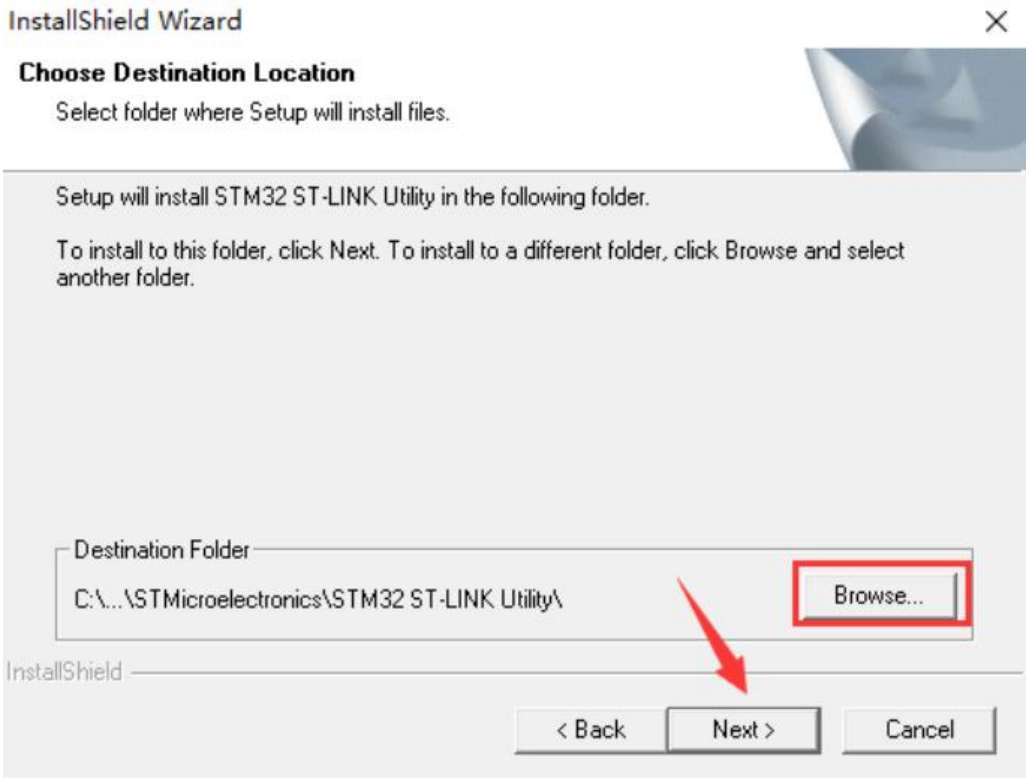
Step 2 In the welcome window, click **Next**.



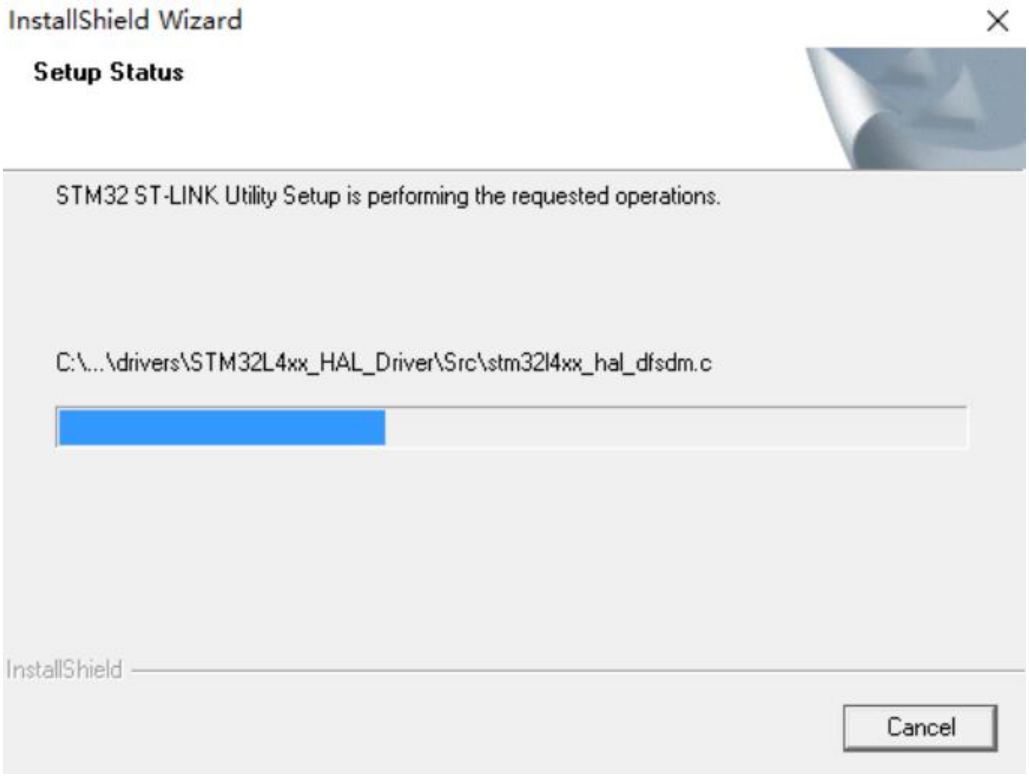
Step 3 In the **License Agreement** window, click **Yes**.



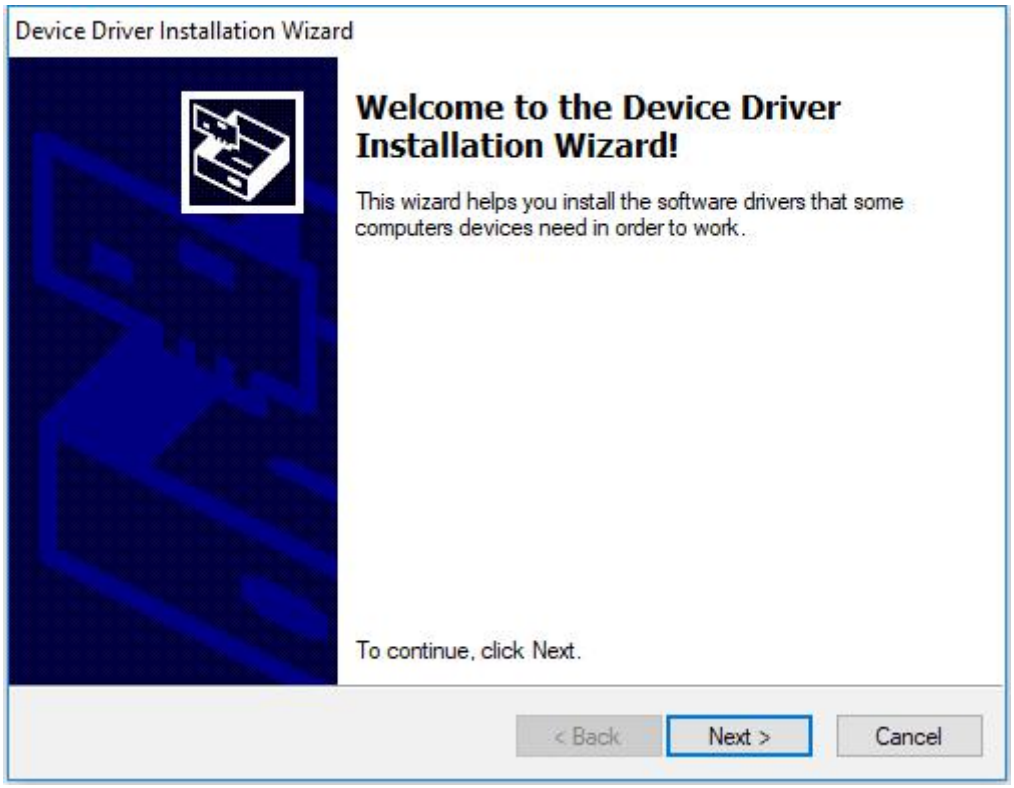
Step 4 In the **Choose Destination Location** window, select an installation path and click **Next**. (The following figure uses the default path.)



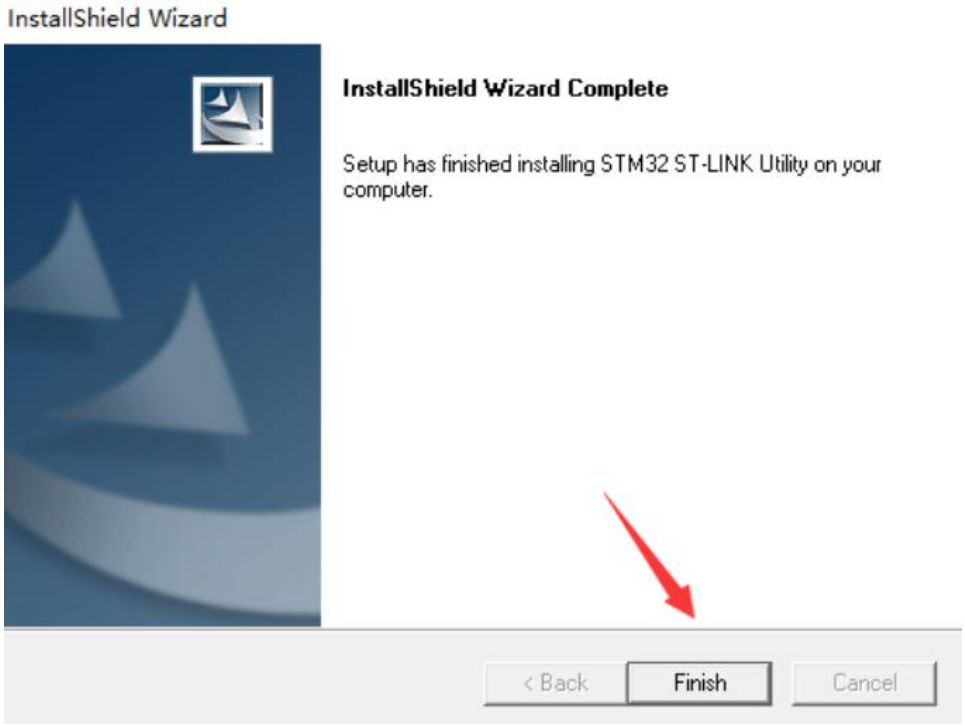
Step 5 Wait for the installation to complete. The process takes less than one minute.



Step 6 In the **Device Driver Installation Wizard** window, click **Next**.



Step 7 Click **Finish** to complete the installation of the STM32 ST-LINK Utility and ST-LINK driver.

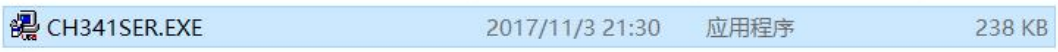


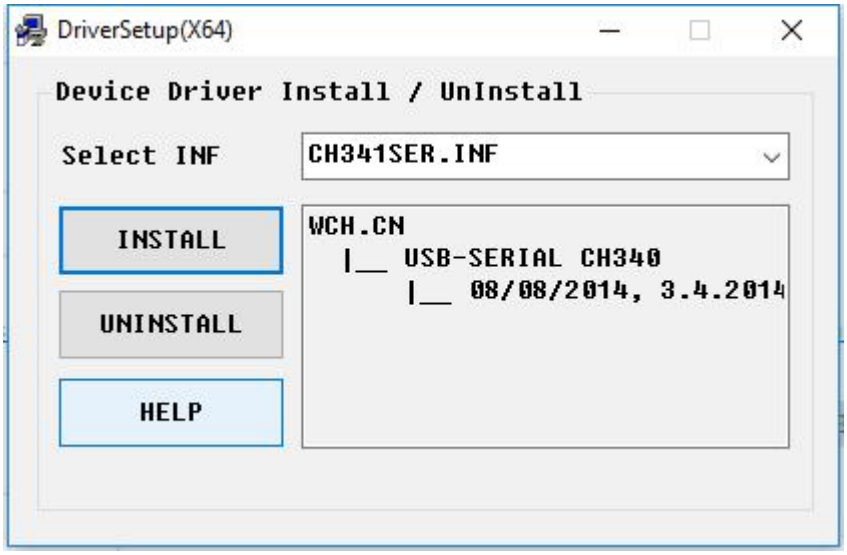
----End

2.3 Installing the NB-IoT Module Debugging Tool

2.3.1 Installing the CH340 Driver

- Driver directory: \EVB_M1\05 Driver
- Installation method: Open the driver installation program and click **Install**.



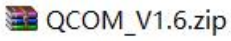


NOTE
If the installation fails, click **UNINSTALL** and then **INSTALL** again.

2.3.2 Installation and Usage of the Serial Port Debugging Assistant QCOM

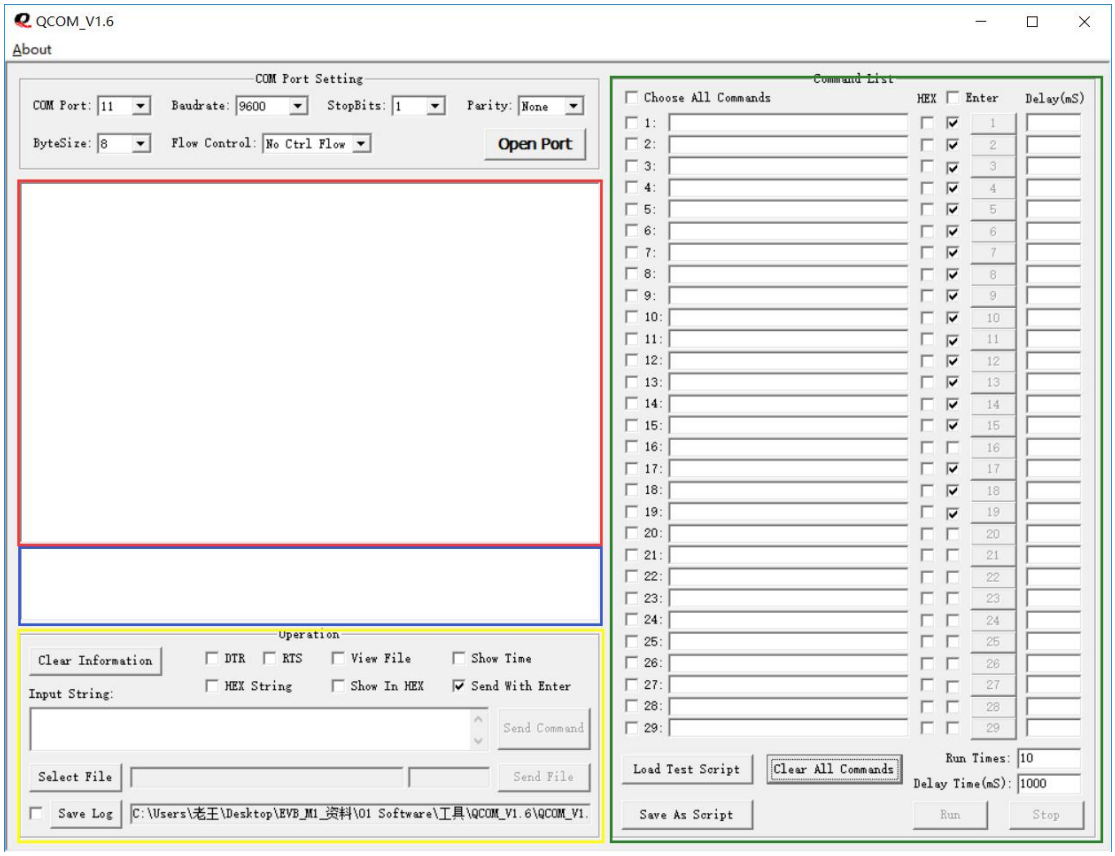
2.3.2.1 Installing the QCOM

- Tool directory: \EVB_M1\01 Software\tools
- Installation method: The QCOM is free of installation. You can use it directly after decompressing the package.



2.3.2.2 Introduction to the QCOM

1. **QCOM description**

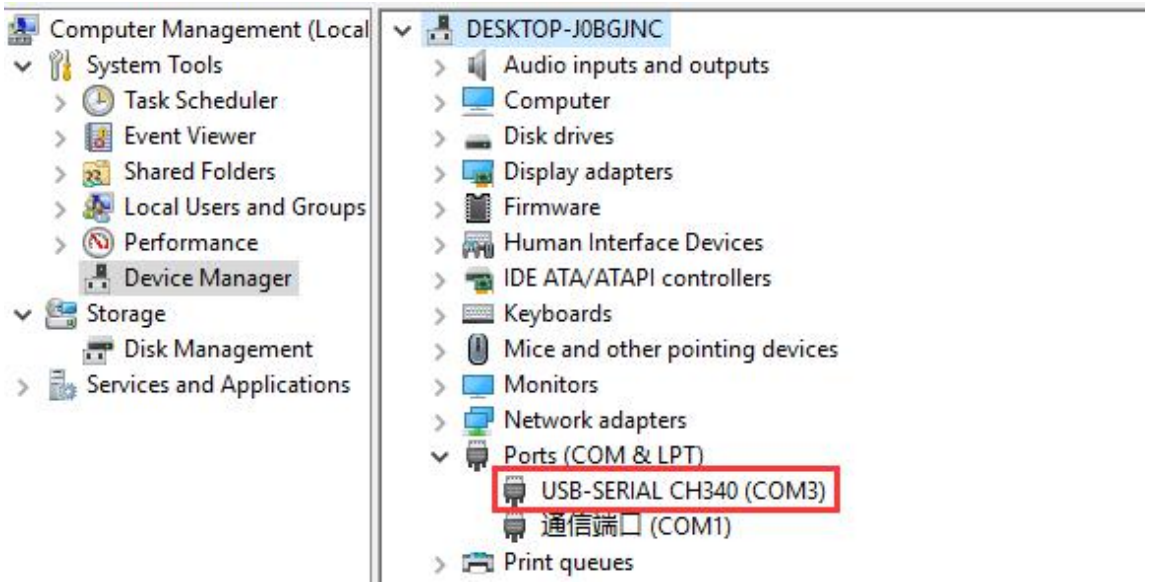


- The red area displays received data.
- The blue area displays status information.
- The yellow area is used to send data or files.
- The green area is used to continuously send data.

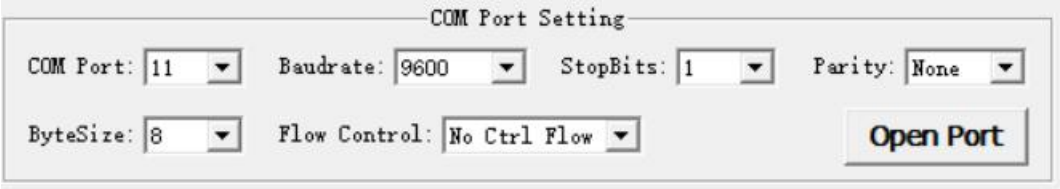
2. COM port settings

Step 2 Select a correct serial port according to the connection between the PC and end device.

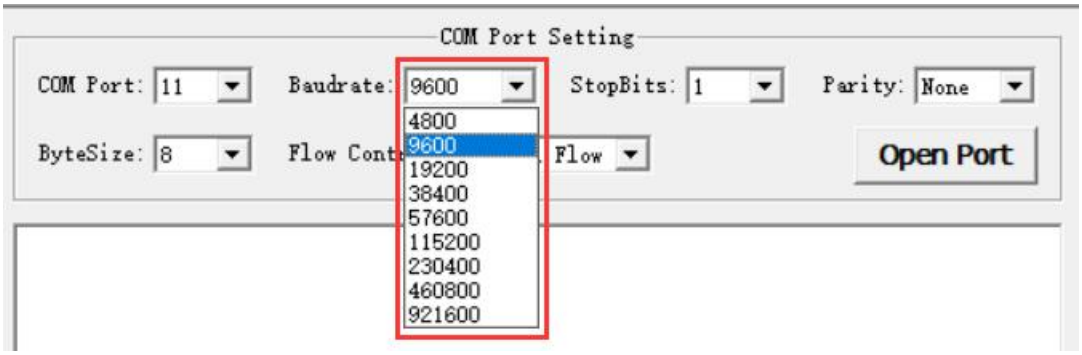
Open the **Device Manager** window. Under **Ports (COM & LPT)**, you can see the port number used by the PC to connect to the development board.



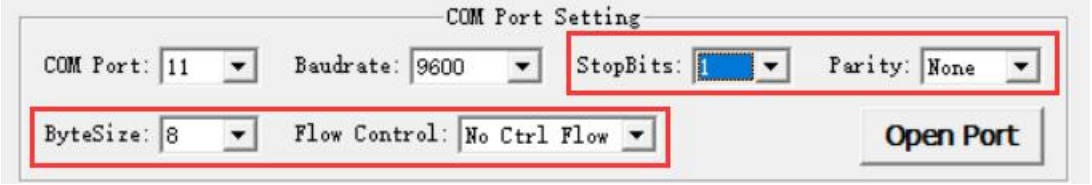
In this example, COM11 is selected.



Step 3 Select a proper baud rate. The default baud rate of the onboard module of the development board is 9600.

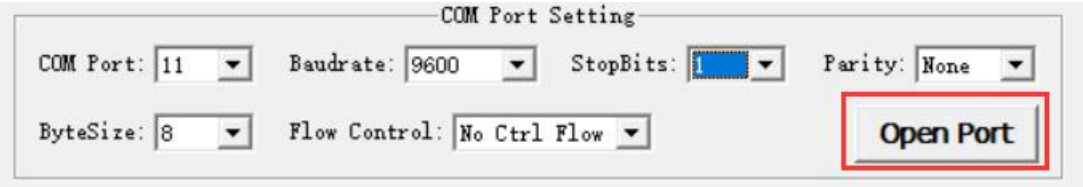


Step 4 Retain the default values for other parameters.

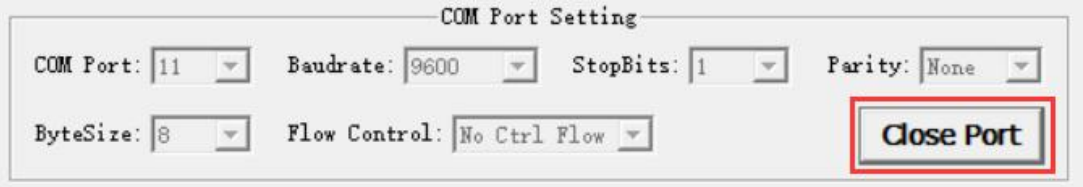


3. COM port opening and closing

- Click **Open Port** to open the selected COM port.

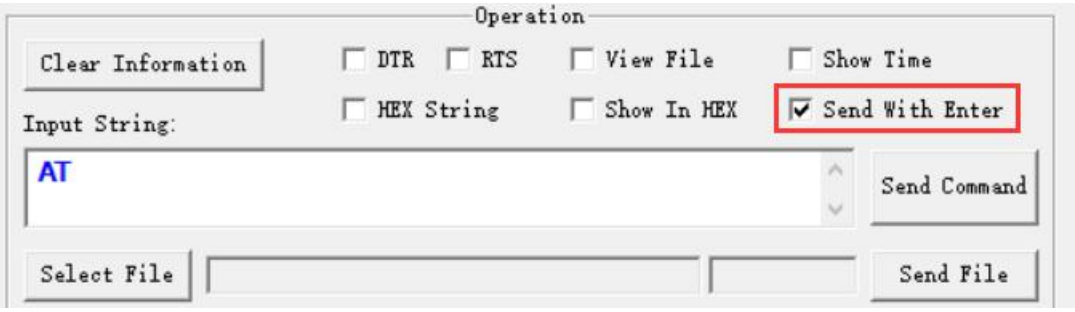


- Click **Close Port** to close the selected COM port.

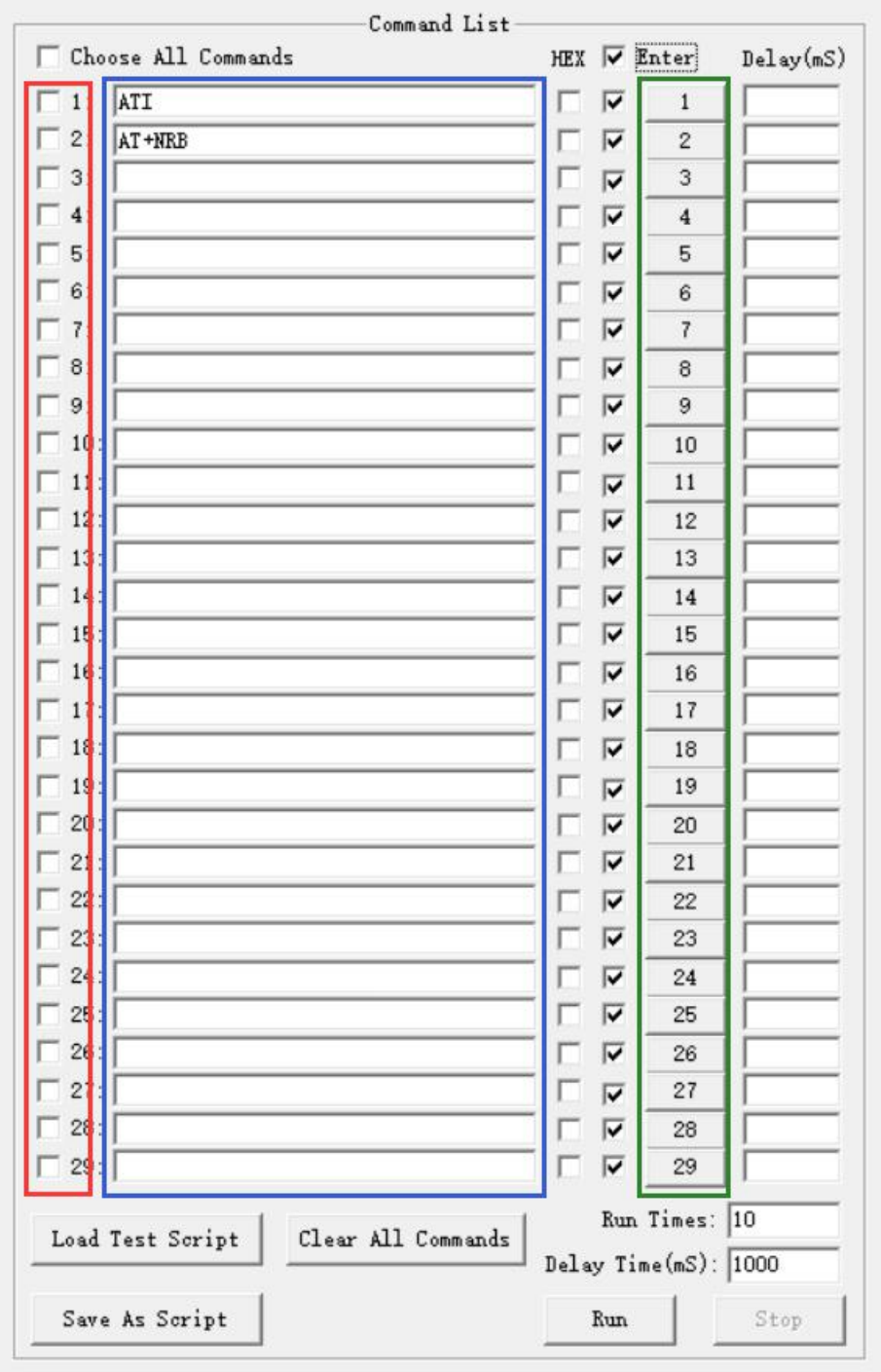


4. Data transmission

In the data transmission window, enter the **AT** command to interact with the module. **Send with Enter** must be selected.



5. Continuous data transmission



The areas and fields are described as follows:

- The red area is used to send data.
- The blue area is used to enter the data to be sent.
- The green area is used to start data transmission.
- **Choose All Commands**: Enables all available commands, which will be sent.

- **HEX**: Indicates that a character string to be entered is a hex-character string.
- **Enter**: Sends data.
- **Delay**: Indicates the delay time of each data.
- **Delay Time**: Indicates the default delay time.
- **Run Times**: Indicates the number of times that all selected data is sent continuously.
- **Run**: Starts to send all selected data continuously.
- Stop: Stops sending all selected data.
- **Save As Script**: Saves all data and configures it as an .ini file.
- **Load Test Script**: Loads data and configures it in the .ini file.

----End

2.4 Obtaining IoT Platform Resources

2.4.1 Obtaining the Huawei IoT Platform

The trial account of Huawei IoT platform in Europe is still manually applied for. Specifically, fill in the following table and send it to Zhou Weimin (weimin.zhou@huawei.com) or Cao Peng (naruto.cao@huawei.com).

Company Name	
Project Name	
Name of the Applicant	
Position of the Applicant	
Telephone of the Applicant	

Please fill in real information so that the staff of the developer community can contact you in time to solve the problems occurred in the environment and interconnection with the Huawei IoT platform.

We will keep your information confidential and do not disclose it.

3 Practice

3.1 Experiment on Turning on the LED Indicator

3.1.1 Objective

The preceding sections describe the installation methods of the driver and corresponding development software as well as their usage. In this section, a simple routine test is performed to turn on the onboard LED indicator.

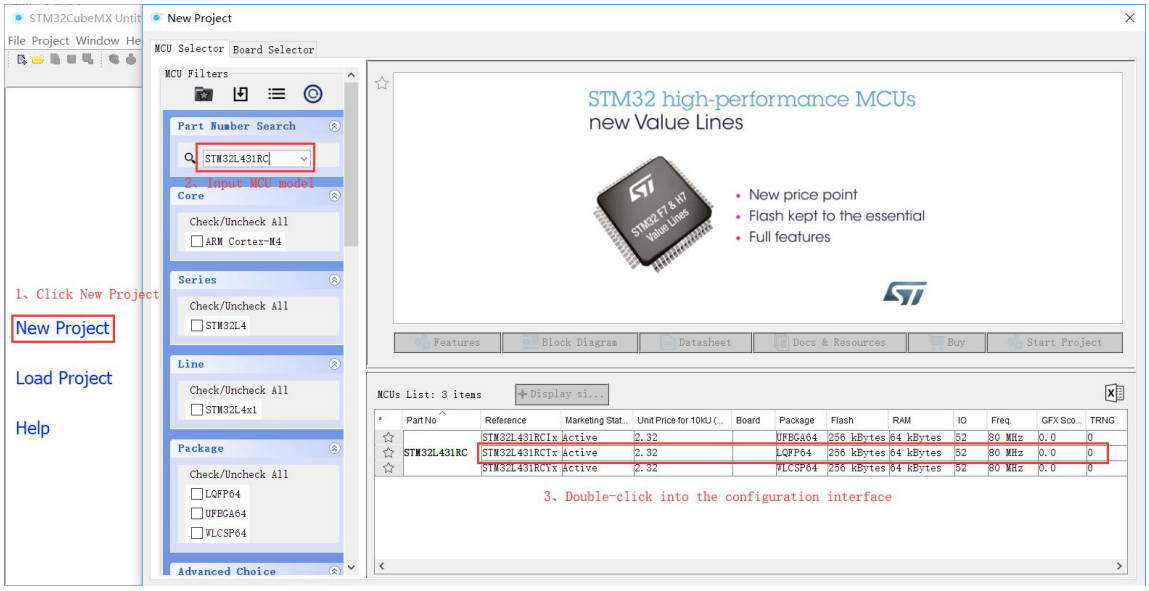
3.1.2 Experiment Design

The LED indicator on the EVB_M1 development board is connected to the PC13 pin of the MCU. When the output of the pin is high level, the LED indicator is on.

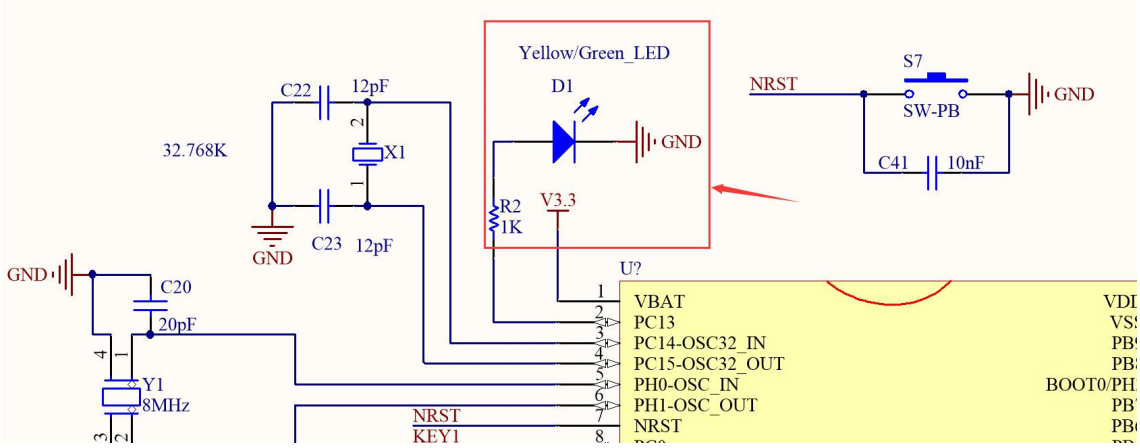
3.1.3 Procedure

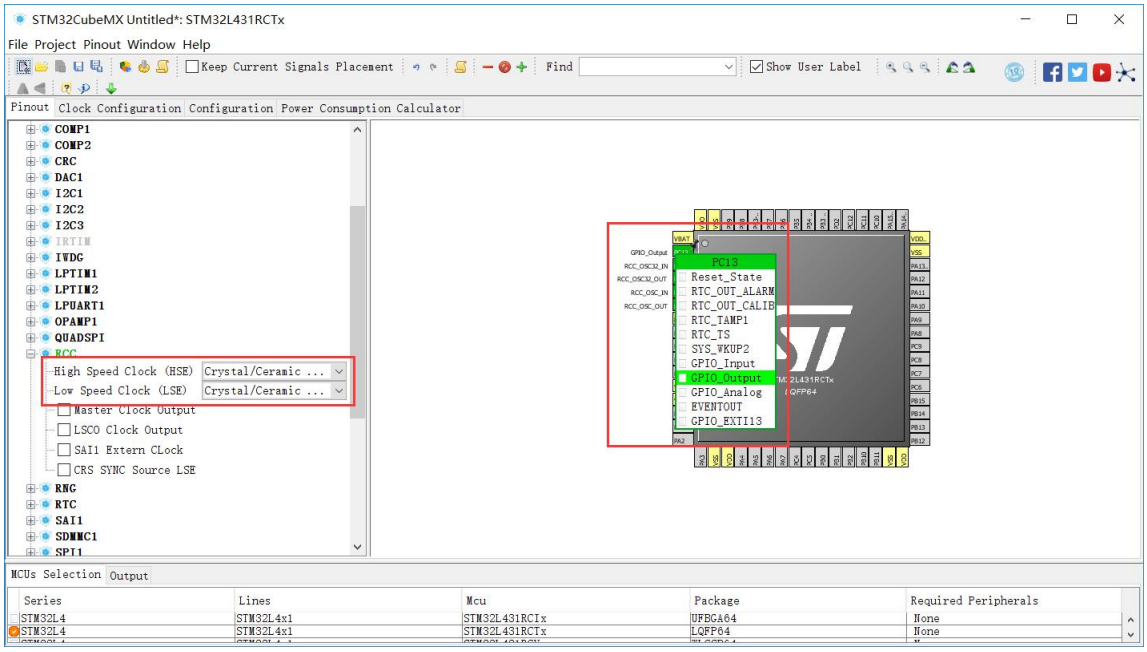
3.1.3.1 Creating a Project by Using STM32CubeMX

Step 1 Double-click STM32CubeMX to open it. Click **New Project**. In the displayed MCU selection box, enter **STM32L431RC**. In the right pane, select the model with the package type of LQFP. Then, double-click it to access the configuration GUI.

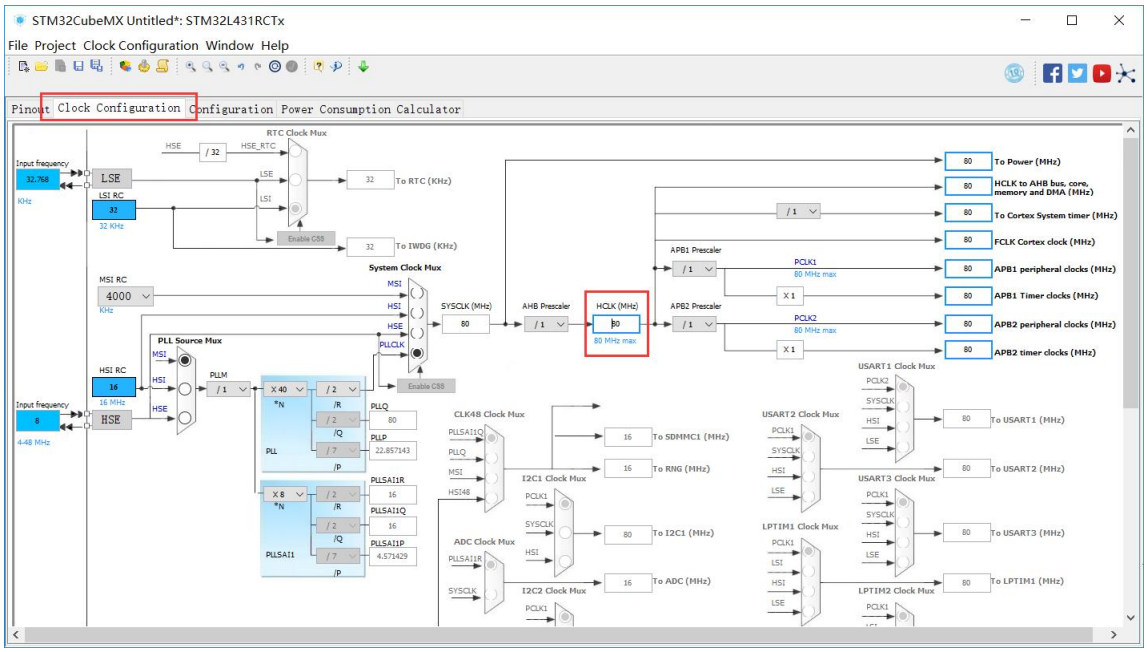


Step 2 Open the schematic diagram of EVB_M1. It is found that EVB_M1 has high-speed and low-speed external crystal oscillators. The LED indicator is connected to the PC13 pin. When the PC13 output is high, the LED indicator is on. Therefore, the external crystal oscillator is preferentially configured on the pinout configuration interface. Under **RCC**, set the high-speed clock and low-speed clock to **Crystal/Ceramic Resonator**, and then right-click the PC13 pin and choose **GPIO_Output** from the shortcut menu.

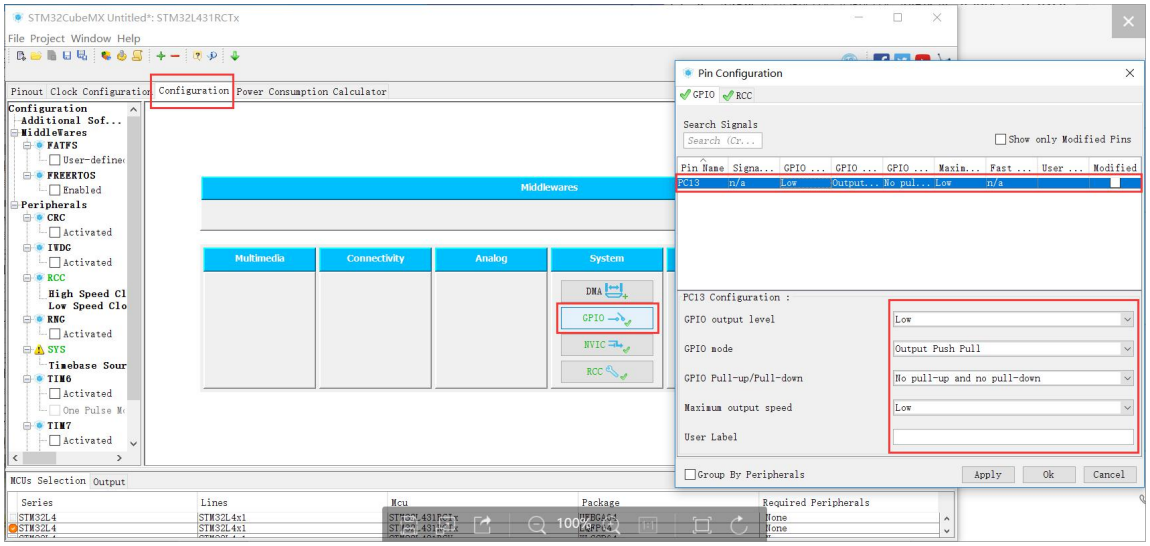




Step 3 Configure the clock according to the project requirements. The following takes the automatic setting as an example. The last frequency is automatically set to **80 MHz** by STM32CubeMX.



Step 4 Access the resource function configuration window. NVIC is not described in this section. The RCC column does not need to be modified. DMA is not used and therefore does not need to be modified. In the **SYSTEM** column, click **GPIO** to access the configuration window. Select the PC13 pin and set initialization parameters. For example, set **GPIO output level** to **Low** (indicating that the LED indicator is off during initialization), **GPIO mode** to **Output Push Pull**, and **GPIO Pull-up/Pull-down** to **No pull-up or pull-down**.



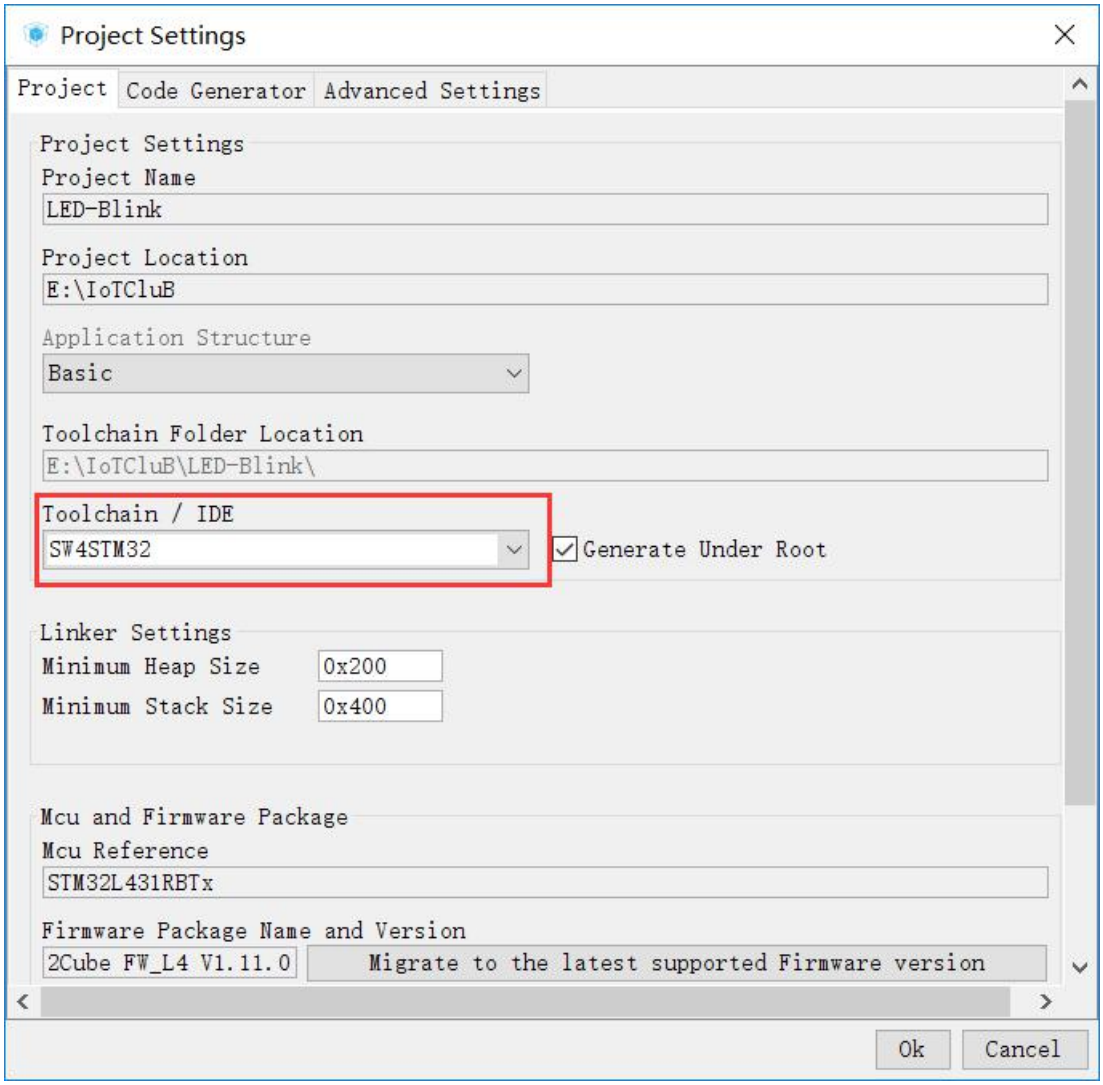
There are two GPIO output modes: output push-pull and open drain (OD) output. (The multiplexing output mode is not described here.)

Push-pull output mode: When high level is output, the P-MOS transistor is conducted. When low level is output, the N-MOS transistor is conducted. The two transistors are conducted alternately. One is responsible for importing the current and the other is responsible for pulling the current so that the load capacity and switch speed are much higher than those in the common way. The low level of the push-pull output is 0 V, and the high level is 3.3 V.

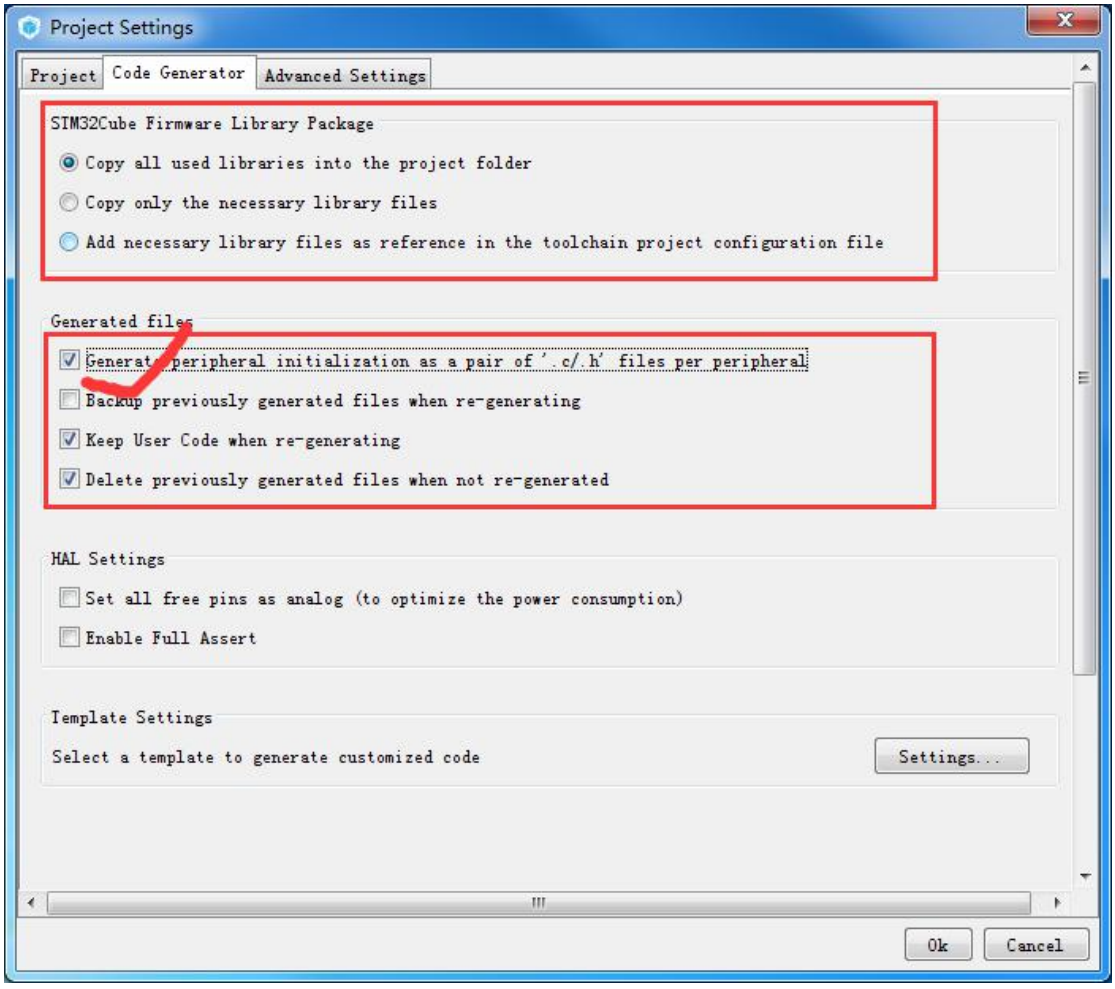
OD output mode: If the output is **0**, low level is output. In this case, conduct the N-MOS transistor and connect its output to the GND. If the output is **1**, neither a high-level nor low-level signal is output. The pin is in high-impedance state. To enable the pin to work normally, an external pull-up resistor must be connected. When multiple pins in OD mode are connected, the pull-up resistor provides a high level only when the output of all pins is in high impedance state. The high level voltage is the voltage of the power supply connected to the external pull-up resistor. If any of these pins is low level, the circuit is equivalent to be short-circuited to the GND and the entire circuit is low level (0 V).

Generally, the push-pull output mode applies to scenarios where the output level is 0 V or 3.3 V. The OD output mode is used when the levels do not match. If the 5 V high level needs to be output, an external pull-up resistor needs to be connected. The power supply is 5 V, and the GPIO is set to the OD mode. When the output is in high impedance state, the pull-up resistor and the power supply output the 5 V level.

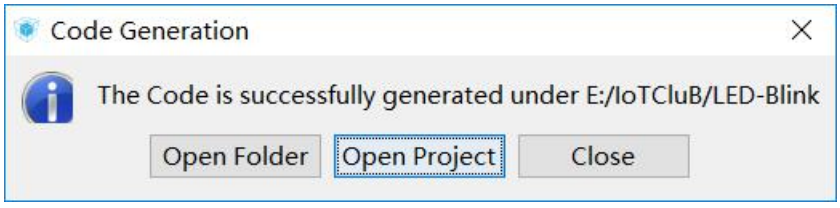
Step 5 Click the screw icon on the toolbar to perform configuration for generating code. On the **Project** tab of the **Project Setting** page, set the project name. The project path has been specified in the preceding section and is not modified here. SWSTM32 is selected as the development tool for generating a project.

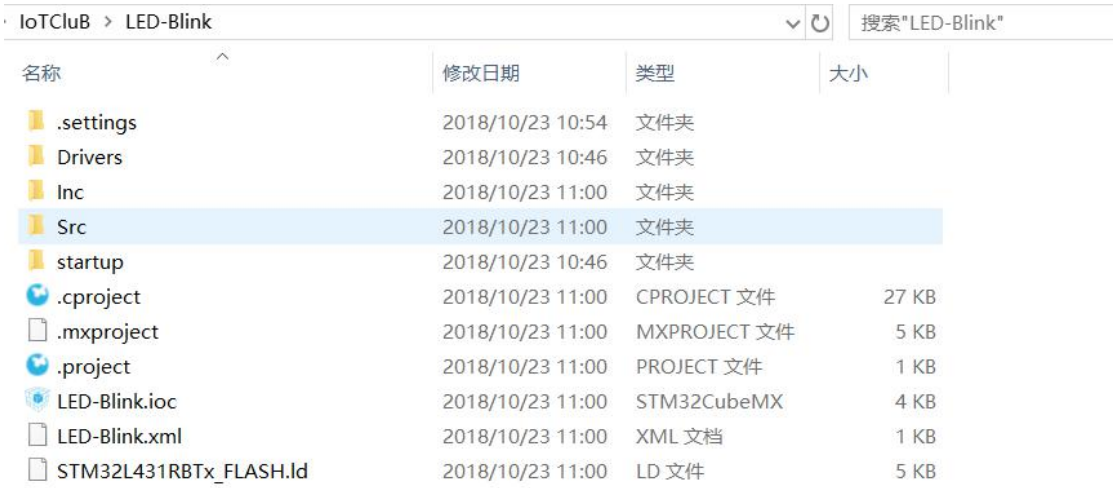


Parameters on the **Code Generator** tab also need to be configured. Select **Generate peripheral initialization as a pair of '.c/.h' files per peripheral** in the **Generated files** area, so that the generated project will place the initialization code of the resources (IIC, SPI, USART, GPIO, and so on) in the corresponding .c file. This facilitates data adding, deletion, modification, and query.



Step 6 Click **Ok** to generate the project code. You can open either the project folder or the project directly. In this example, the project folder is opened. The project folder is divided into two parts: project directory of SW4STM32 and project of STM32CubeMX. You can also use STM32CubeMX to open the STM32CubeMX project file to add, delete, or modify the project.





----End

3.1.3.2 Project Directory Structure

- The directory structure of the project is as follows:

  |——Drivers

  |——CMSIS (This is a microcontroller peripheral standard defined by ARM. This folder contains kernel, DSP, RTOS files and files of peripherals such as USB and SDIO.)

  |——STM32L4xx_HAL_Driver (peripheral driver of the STM32L4xx hardware abstraction layer)

  |——Inc (The **include** folder generally stores the header file, which is usually a .h file.)

  |——Src (The **source** folder stores the source file, which is generally a .C file.)

  LED-Blink.ioc and .C file both are STM32CubeMX files. .C file is automatically generated, while you can double-click **LED-Blink.ioc** to open it using the STM32CubeMX for later configuration.

  The following table describes the files included in the **STM32L4xx_HAL_Driver** file.

File	Description
stm3214xx_hal_ppp.c/.h	Contains APIs of the basic peripheral. **ppp** indicates any peripheral, such as IIC, SPI, and USART.
stm3214xx_hal_ppp_ex.c/.h	Contains APIs for expanding peripheral features.
stm3214xx_hal.c/.h	Contains HAL generic APIs, such as HAL_Init, which should be included in the developer's code.
stm3214xx_hal_conf.h	This is an HAL configuration file, which should be configured by the developer to select the peripheral to be enabled. The peripheral must be included in the customer's code.
stm3214xx_hal_def.h	Contains the definition of HAL common data types and macro definition.
stm3214xx_11_ppp.c	Contains APIs for implementing bottom-layer functions in some complex peripherals. These APIs are called through **stm3214xx_hal_ppp.c**.

- Project files are as follows:

  The project directory consists of four parts:

  |——Startup file

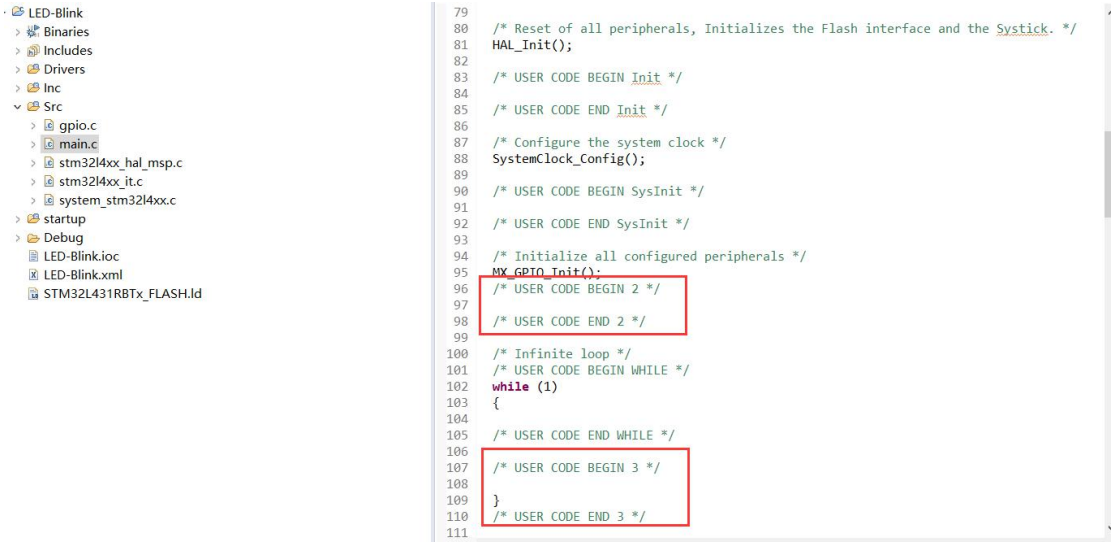
  |——Peripheral driver library

  |——CMSIS system file

  |——Customer code

  In each .C file in the customer code part, there are multiple **BEGIN** and **END** marks, and each **BEGIN** or **END** mark has a corresponding prompt message. Developers only need to place their code in the corresponding area. (This operation is optional, while it shortens the R&D time when STM32CubeMX is used again for development.)

  No error or warning message is displayed after compilation.

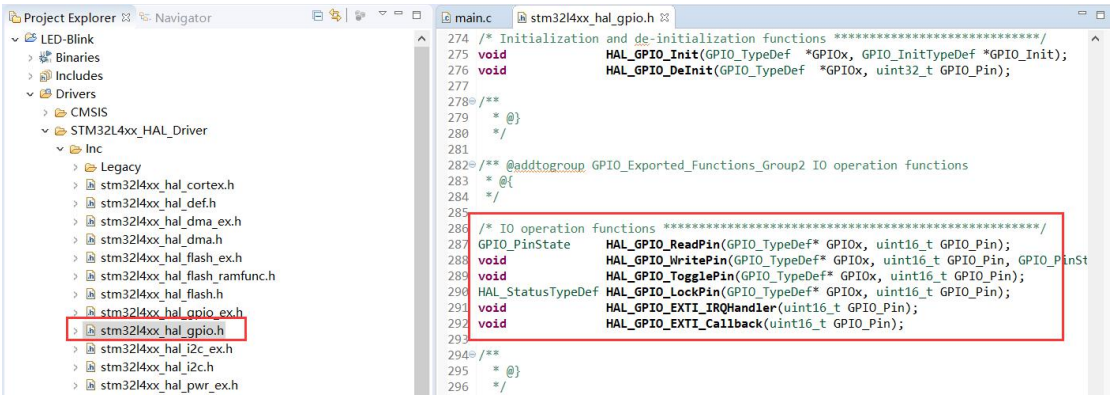


3.1.3.3 Instruction for Developing the Code for Turning on the LED Indicator

- Locate the GPIO interface.

  According to the **main.c** file in the preceding project directory, the project first resets all peripherals, initializes the flash interface and SysTick, and configures the initialized system clock and then GPIO interface. In this example, only the GPIO interface is used. If other components are used, they will also be initialized. The initialization function is provided in the **gpio.c** file. Then, you can develop code for desired applications. In this example, we want to turn on the LED indicator and then make it blink.

  Because the HAL library is encapsulated well, we do not need to deal with the registers at the bottom layer. Instead, we only need to use the interfaces provided by the peripheral library. To turn on the LED indicator, you need to pull up or down the corresponding pins, which can be implemented by calling the required interface function is in the **stm32l4xx_hal_gpio.h** file. Specifically, perform the following steps: Expanding **Drivers/STM32L4xx_HAL_Driver/Inc** folder and double-click **stm32l4xx_hal_gpio.h** to open it. Locate **/* IO operation functions **/**. The functions below are GPIO interface functions.



- Execute the code to pull up the PC13 pin and turn on the LED indicator.

  The GPIO interface has the following function:

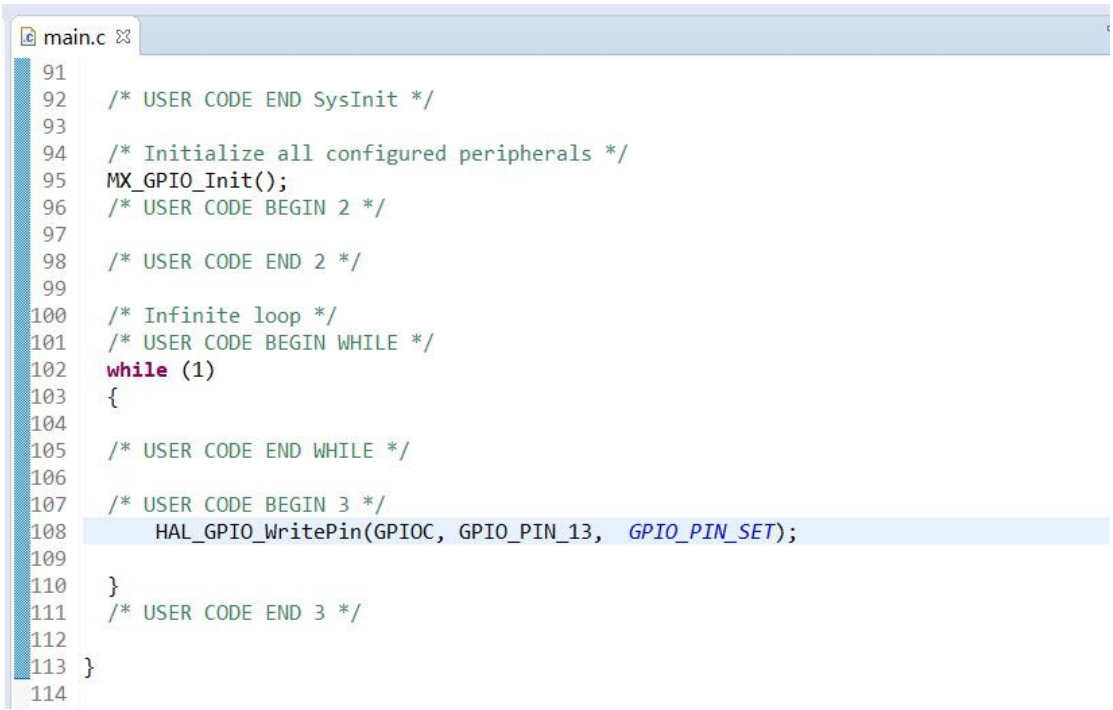
  HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);

  This function is used to set whether to pull up or down a pin.

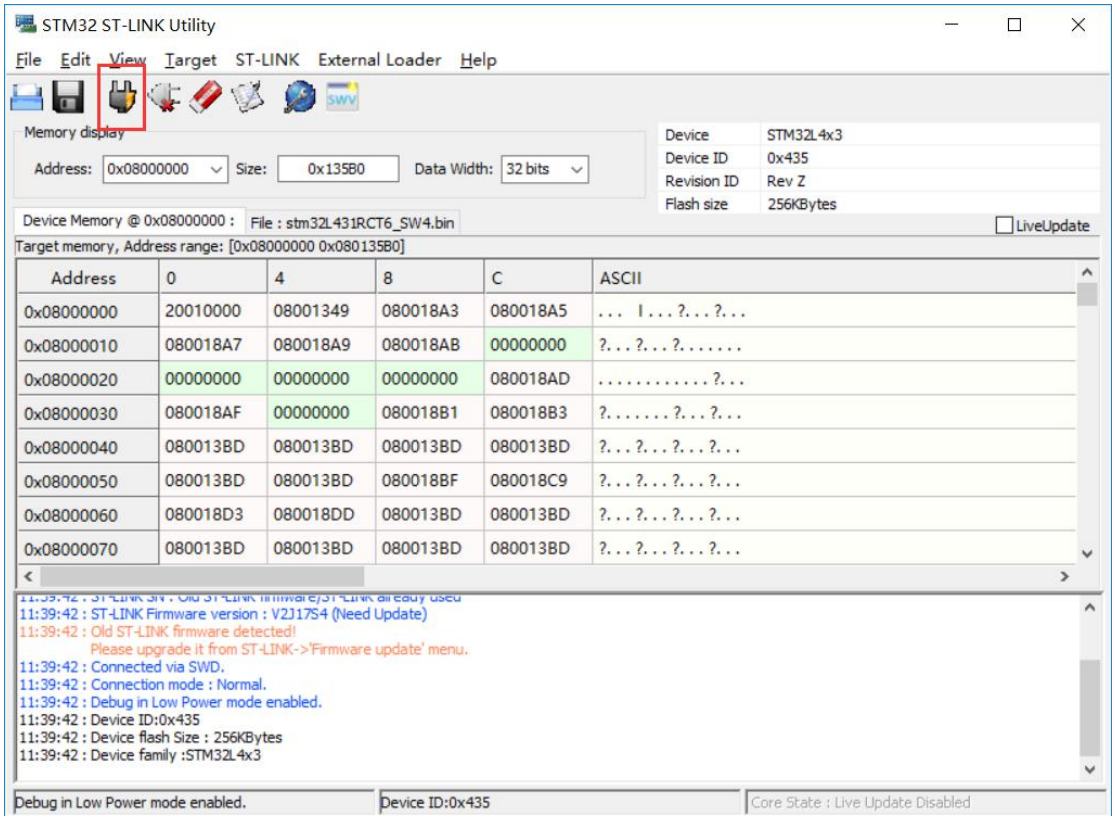
  Copy this function name to the area between **USER CODE BEGIN 3** and **USER CODE END 3** under **while** in the **main.c** file.

  Set parameters in the function according to the prompt message and type and compile code.

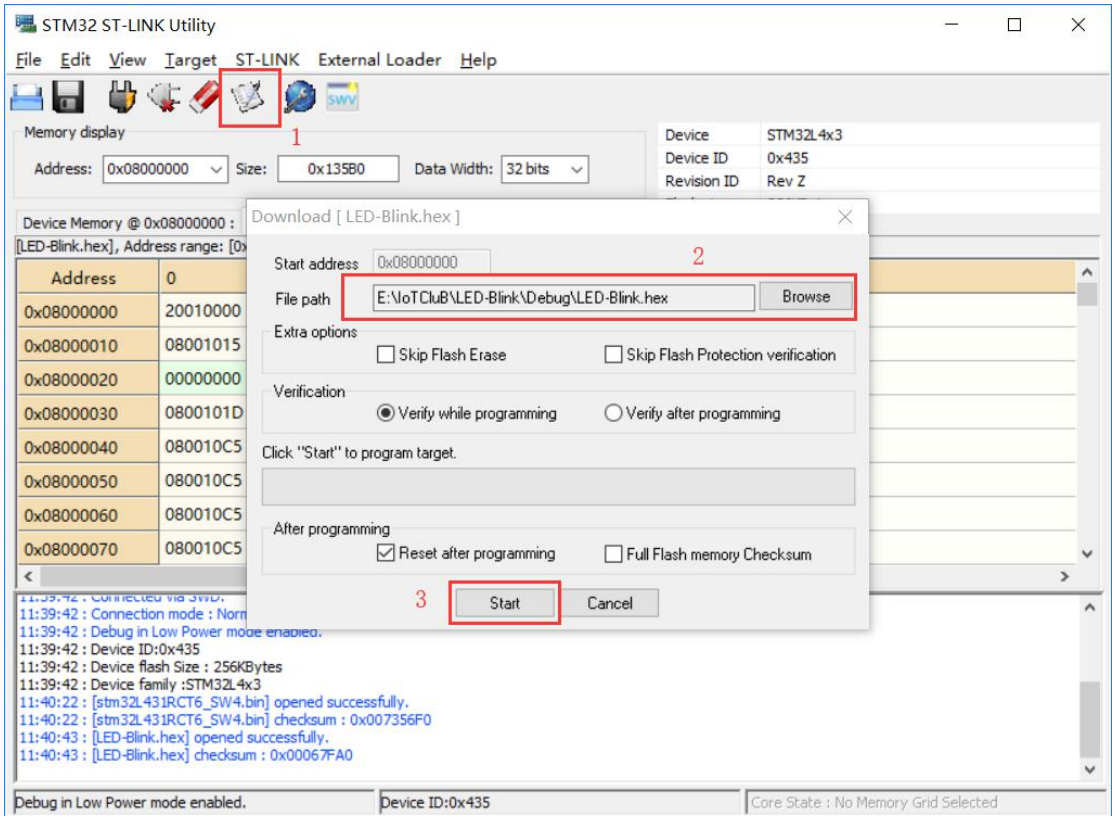
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);



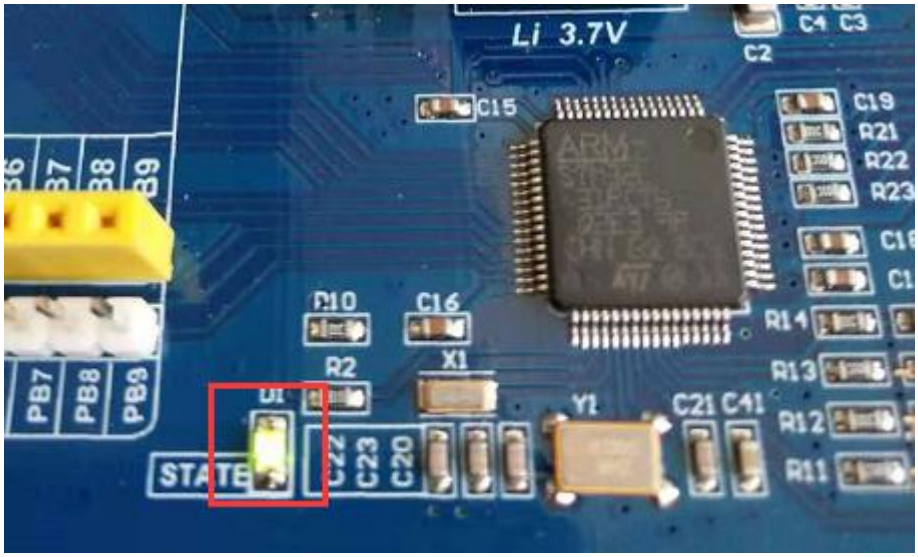
- Open the STM32 ST-Link Utility and connect it to the burner.



- Select the burning file and program.



The following figure shows the result.



3.2 Experiment on the NB-IoT's Access to the Network

3.2.1 Objective

3.2.2 Preparations

1. Hardware preparation
   - Install the antenna, insert an SIM card, and connect a micro USB cable to the NB-IoT by referring to section 2.3 "Installing the NB-IoT Module Debugging Tool."
   - Switch the jumper cap connection, as shown in the following figure. Then power on the board so that the AT command can be sent to the debugging module through the serial port assistant of the computer.

2. Software preparation
   - Install the CH340 serial port driver by referring to section 2.3
   - Open the serial port assistant by referring to section 2.3 and enable the serial port.

3.2.3 Procedure

By default, the NB-IoT module is automatically connected to the network. After the module is powered on, you only need to send the following commands and wait for the module to automatically connect to the network:

Command 1: AT+NRB

This command is used to restart the module.

Command 2: AT+CFUN?

This command is used to check whether the card circuit is ready. After the module is restarted or powered on again, wait for 20 seconds and then send this command. If **1** is returned, the card circuit is ready. If **0** is returned, the card may not be connected to the module. Then, you need to check whether the card is inserted properly or whether the card slot is damaged.

Command 3: AT+CSQ

This command is used to query the signal strength. The first parameter returned by this command indicates the signal strength. The value range is 0–31. A larger value indicates a higher signal strength. The value of the second parameter returned is always **99**.

Command 4: AT+CGATT?

This command is used to check whether the module is attached to a base station. If **1** is returned, the module is attached to a base station, indicating that data can be transmitted and received. If **0** is returned, the module is not attached to a base station yet.

3.3 Connecting the NB-IoT to Huawei OceanConnect

3.3.1 Objective

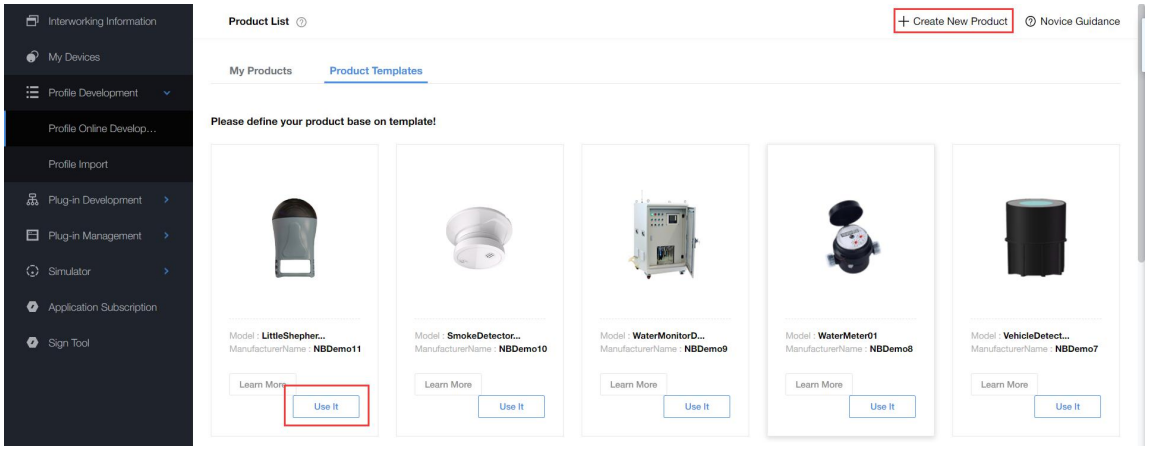
The CoAP test requires a cloud platform. This section uses the Huawei OceanConnect platform as an example. The following describes the platform application, platform-side development, and the test on data sending and receiving between the EVB_M1 and platform.

3.3.2 Developing a Profile

3.3.2.1 Adding a Product

The profile of a device describes the device type and service capabilities. It defines the service capabilities, attributes, and commands of devices of the same type. To add a product, perform the following steps:

Step 1 Log in to the developer portal and choose **Profile Development** > **Profile Online Development** from the navigation tree on the left. On the **Product List** page, click **Custom Product** in the upper right corner. On the displayed **Product Templates** tab, you can use a template to define your product. Specifically, click **Use It** in the lower right corner of the product template. The parameters are defined based on the specific device.

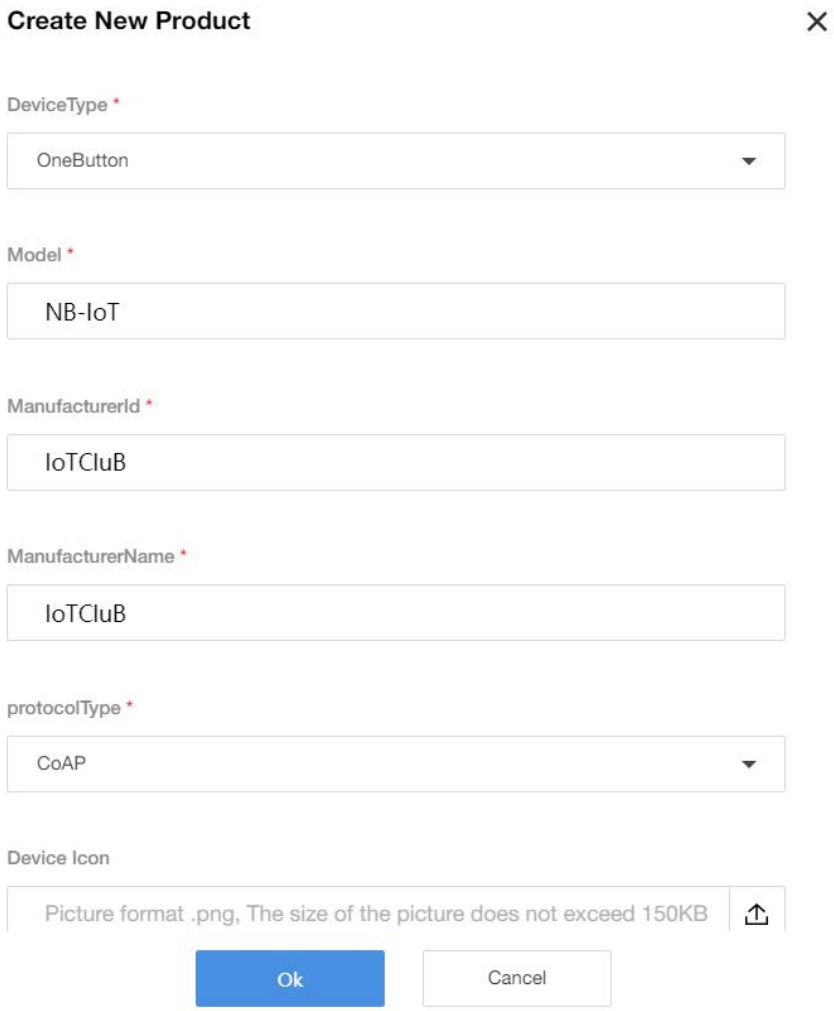


Alternatively, you can click **Create New Product** in the upper right corner to define a product. The following uses creating a product as an example.

Step 2 Set parameters based on actual situations, such as the device type, device model, manufacturer ID, and manufacturer name. Click **Ok** to complete product adding.

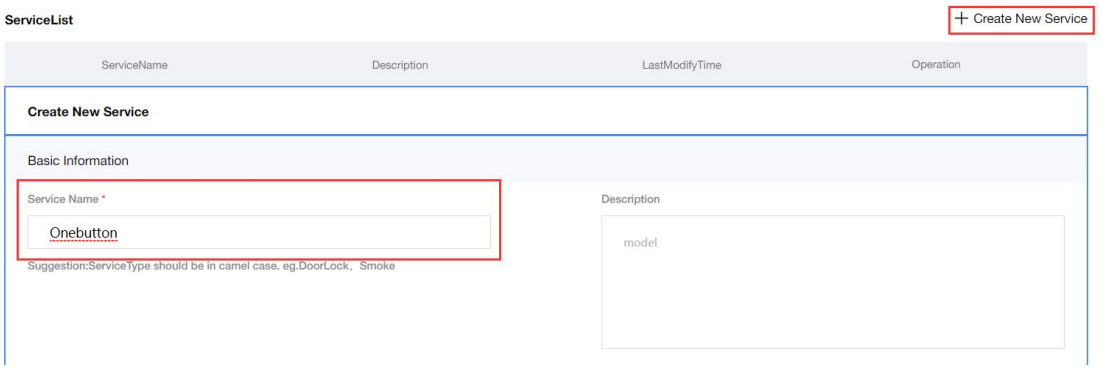
DeviceType: Indicates the device type. You can select a value from the drop-down list. **OneButton** is used as an example in the following figure.

Model: Indicates the device model. **NB-IoT** is used as an example.
ManufacturerId: Indicates the manufacturer ID of the device. **IoTCluB** is used as an example.
ManufacturerName: Indicates the manufacturer name of the device. **IoTCluB** is used as an example.



Step 3 On the product details page, click **Create New Service** to add basic information, attributes, or commands as prompted. Then, click **Save**.

Add basic service information. The service name is in camelCase format. Here, **OneButton** is used as an example. The description is optional. If there are multiple services, add multiple services.



----End

3.3.2.2 Adding an Attribute

AttributeName: Indicates the attribute name. **Button** is used as an example.

DataType: Indicates the data type. The value is **int**, **float**, **datetime**, **string**, **jsonObject**, and **array**. **int** is used as an example.

The format of reported complex data is as follows:

DateTime: yyyyMMddTHHmmssZ, for example, 20151212T121212Z

jsonObject: Customizes the JSON structure. Data that the platform fails to parse will be transparently transmitted.

Min/**Max**: Indicates the minimum value and maximum value, respectively. These parameters are valid only when **DataType** is set to **int** or **float**.

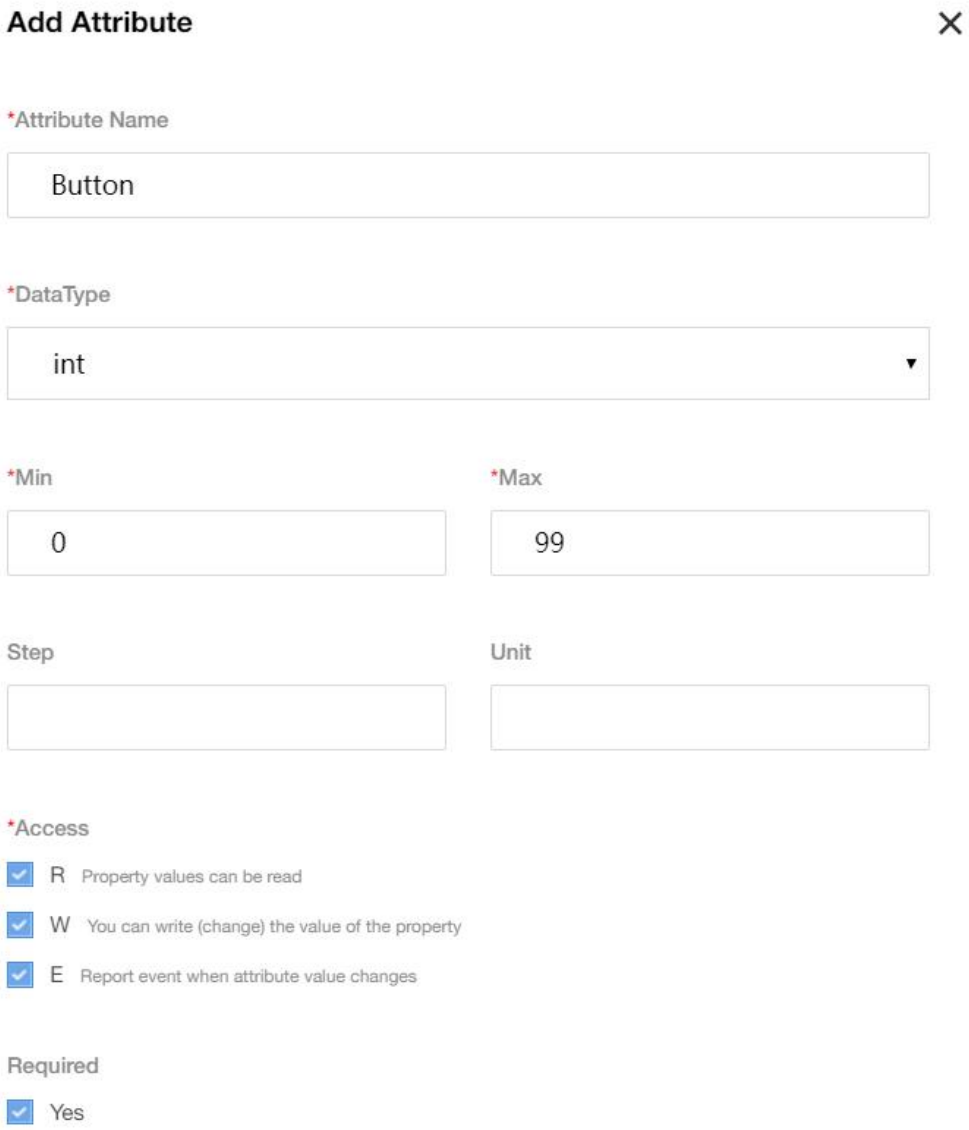
Step: Indicates the step. This parameter is reserved and can be set to **0**.

Unit : Indicates the unit, in English. The value is determined by the attribute. For example, the temperature unit is °C, and the percentage unit is %.

MaxLength: Indicates the length of a character string. This parameter is valid only when **DataType** is **string**, **datetime**, **jsonObject**, or **array**.

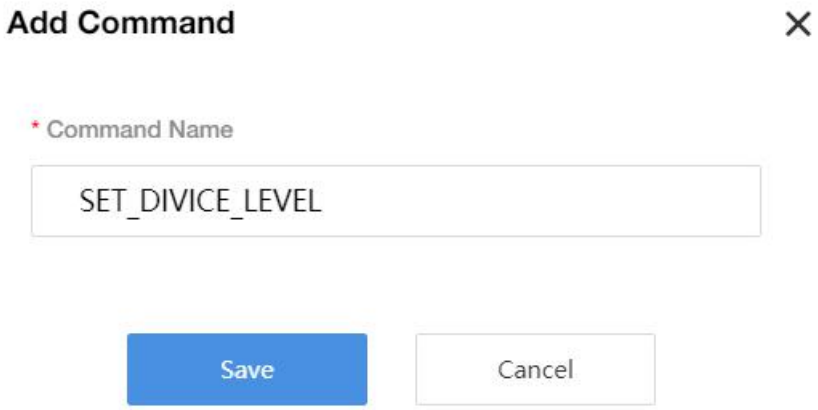
EnumList: Indicate the enumeration list. For example, the value of **activityStatus** can be **STANDBY**, **RECORDING**, or **LIVING**.

Access: Indicates the access mode. The value is **R**, **RW**, **RE**, **RWE**, or null, where **R** stands for readable, **W** stands for writable, and **E** stands for subscribable.

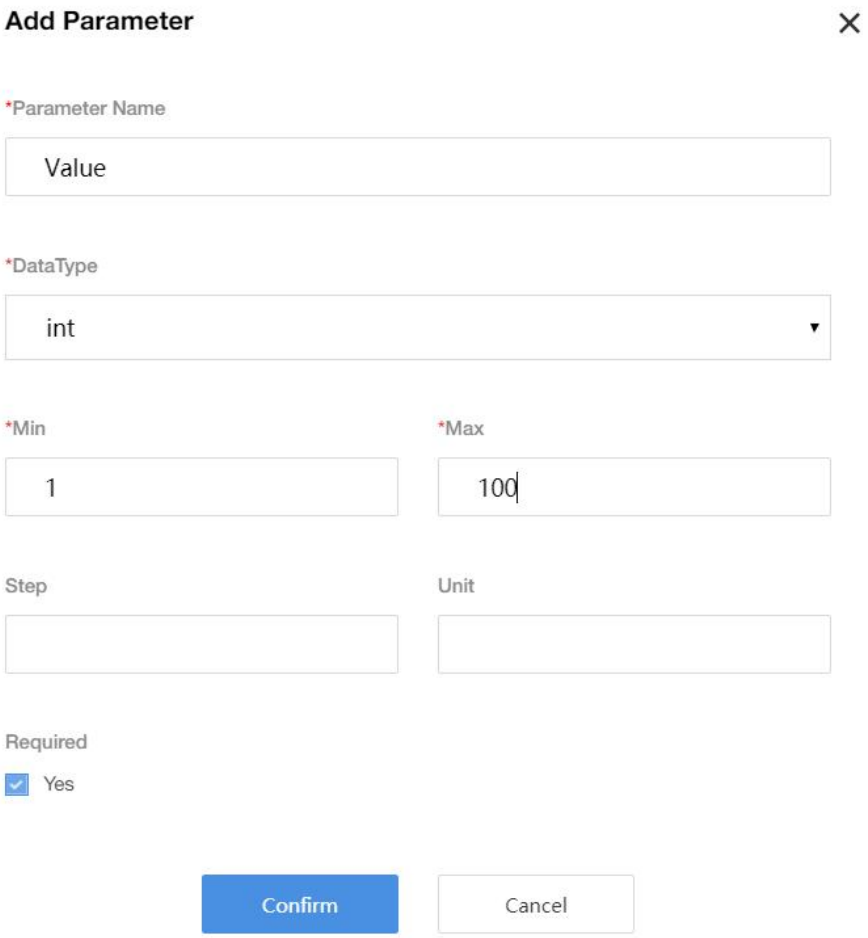


3.3.2.3 Adding a Command

Command name: Indicates the command that can be executed by the device, for example, the **lock** command of the door status sensor and the **VIDEO_RECORD** command of the camera. **SET_DIVICE_LEVEL** is used as an example.



The command name and parameters form a complete command. Click **Add Parameter** on the page.

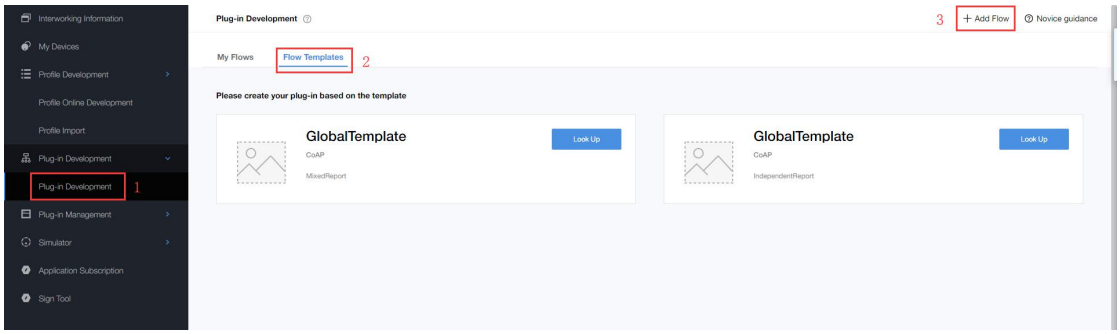


3.3.3 Developing the Encoding and Decoding Plug-in

After developing a profile, you can map messages between the device and the platform in graphical mode on the GUI. The encoding and decoding plug-in template is pre-integrated on the platform. You can select a template based on the device type and access protocol to develop the encoding and decoding plug-in.

Log in to the developer portal and choose **Plug-in Development** from the navigation tree on the left. On the right pane, click **Add Flow** in the upper right corner. On the **Flow Templates** tab, click **Look Up** to view the content of each template.

- If your service is similar to that in the template, you can use the template to modify the plug-in as required.
- If you do not need to use a template, you can create a plug-in. The following describes how to develop an encoding and decoding plug-in. The method is similar to that of using a template to modify the plug-in.



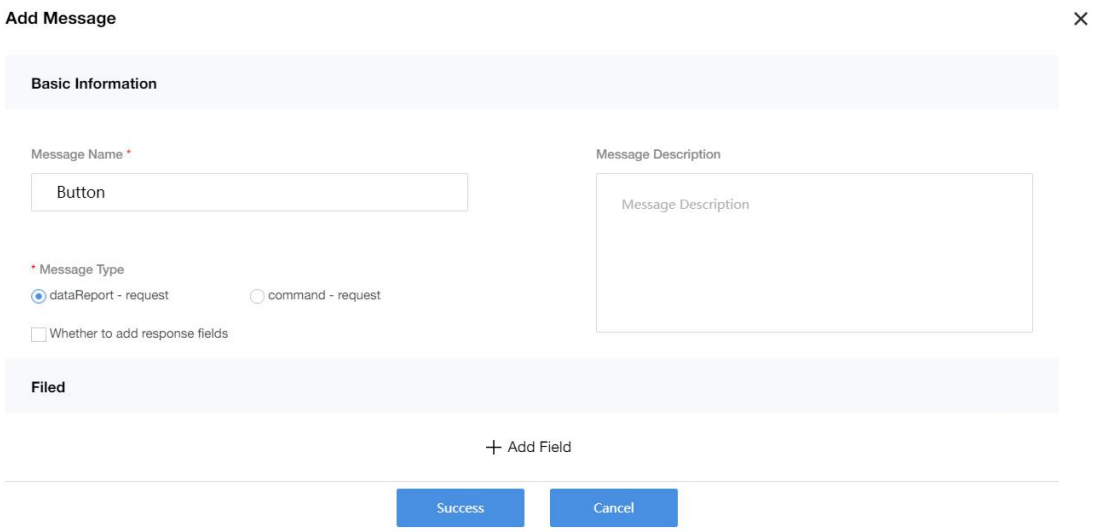
3.3.3.1 Creating a Plug-in

Step 1 Click **Add Flow** in the upper right corner of the page. The plug-in design page is displayed.

NOTE

You can click **Novice Guide** in the upper right corner to view the implementation principle of the plug-in.

Step 2 Add a message. Click **Add Message**, and enter the message name (**Button** is used as an example), description, and message type (including data reporting, command issuing, and response to the command issuing).



Step 3 Add a field. Click **Add Filed** to add a field for reporting data.

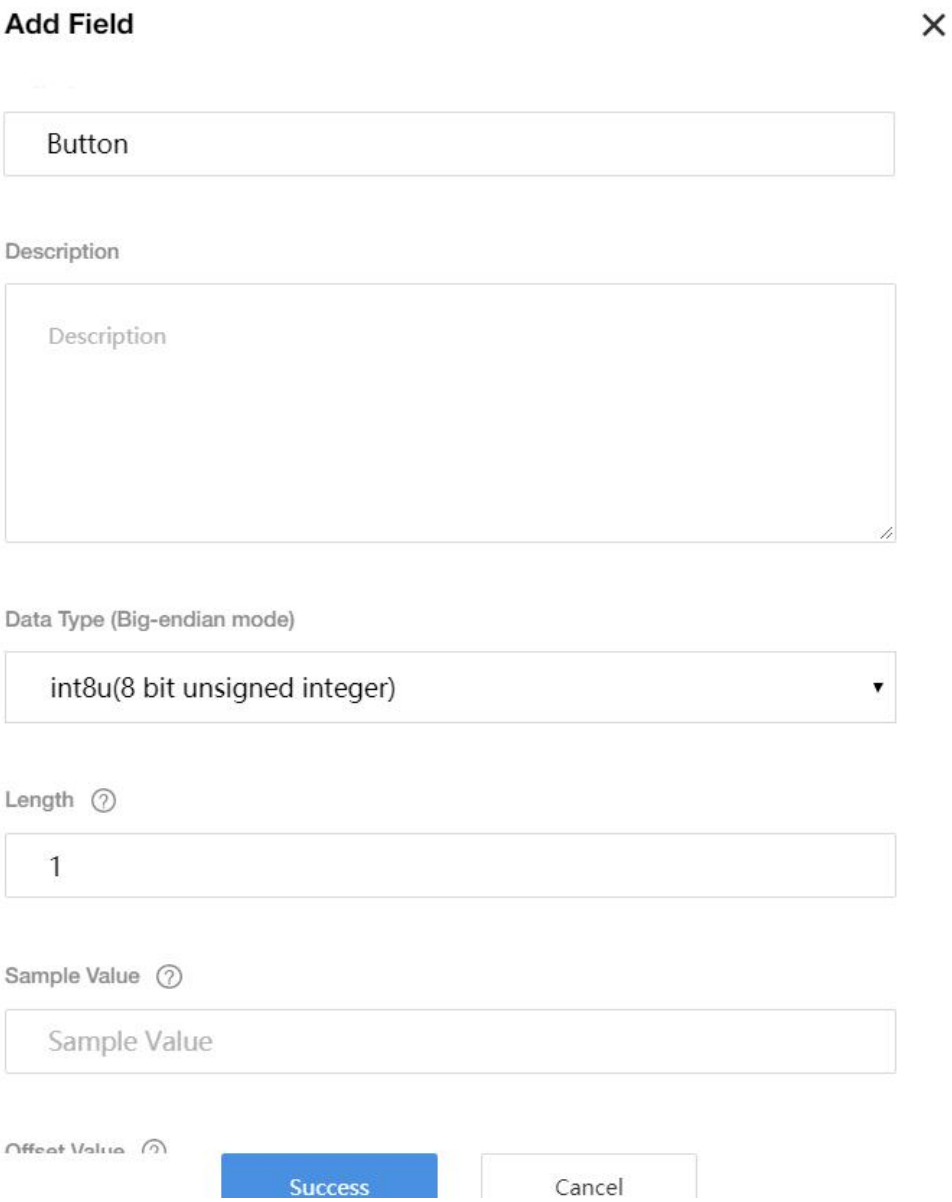
Name: It is recommended that the value be the same as that set in the profile, so that the specified name can map the corresponding field in the profile.

Data type: The value can be **int8u**, **int16u**, **int24u**, **int32u**, **string**, **variablelength string**, **array**, or **variant**. The value maps the **Description** field in the profile.

Length: Indicates the length of the field. If the length is **1**, this field occupies one bit in the reported stream, that is, a hexadecimal value. In this example, the length is **1**.

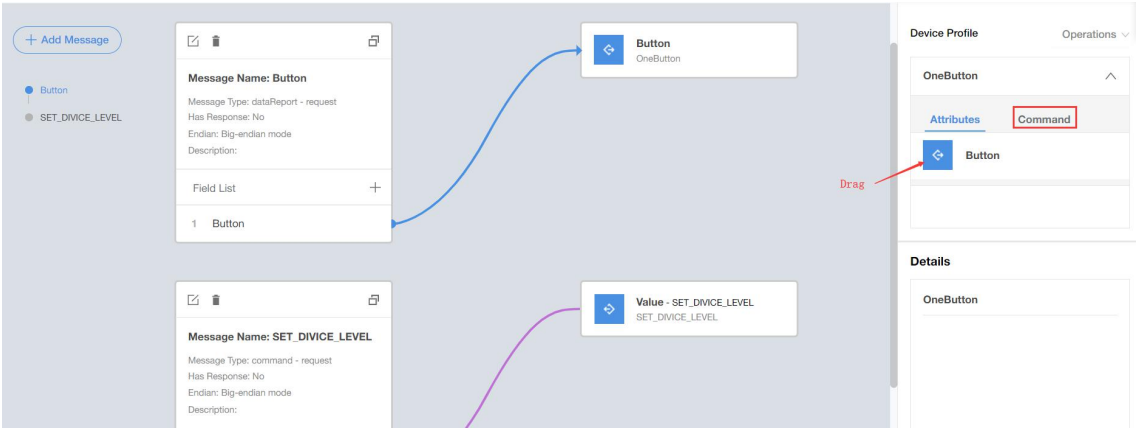
Default value: Indicates the reference value in the stream.

Offset value: Indicates the number of bytes between the current field to the start position of the message stream. For example, if the value is set to **2–3**, this field is located in bytes 2–3 in the stream.



Use the same method to create a command issuing message.

Step 4 Create the mapping between profile attributes, commands, and messages. Design the messages in the plug-in based on the customized profile. Drag attributes or commands in a service to associate them with fields in the message. The attributes correspond to the list of fields in the data report. If there are multiple services, multiple messages should be added. To facilitate understanding, you are advised to set the field name to be the same as the attribute name.



NOTE

The message name for command issuing must be the same as that in the profile.

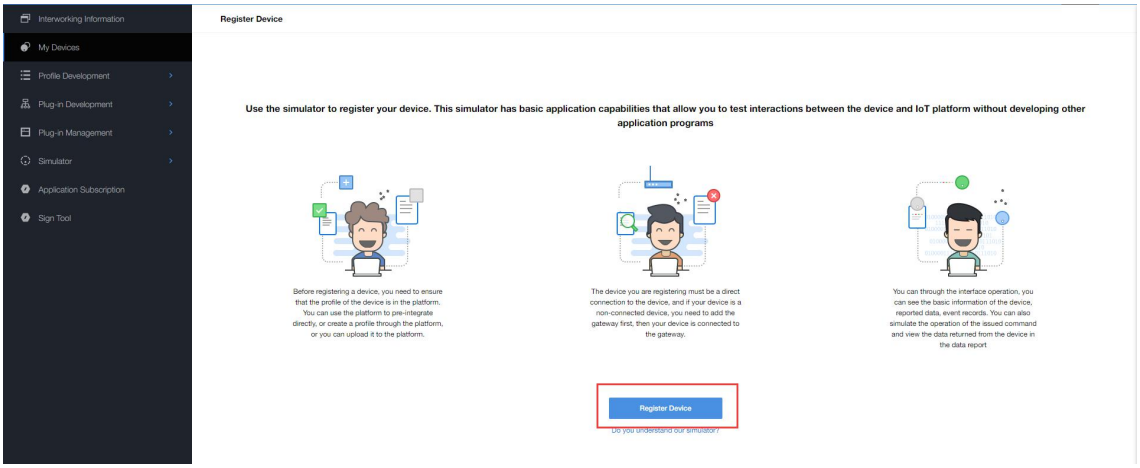
Step 5 After the mapping relationship is created, click **Deploy** at the upper right corner of the page. After the deployment is successful, the system automatically generates an encoding and decoding plug-in package based on your design result.

----End

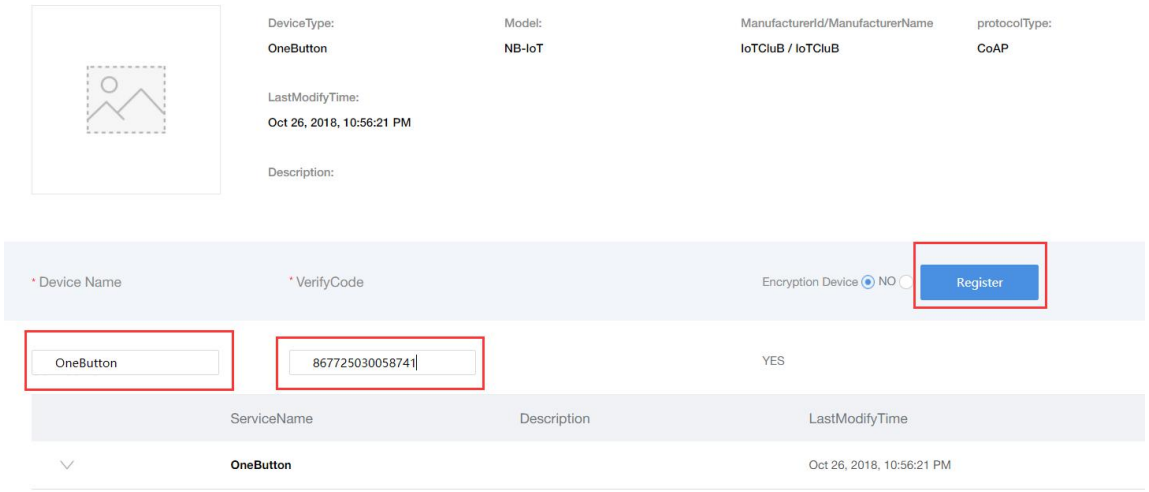
3.3.4 Registering a Device

After the development and deployment are successful, you can connect your device to the IoT platform by performing the following steps:

Step 1 Log in to the developer portal and choose **My Devices** from the navigation tree on the left. On the device list page, click **Register Device** button. The device registration page is displayed. Select the profile for which you want to register the device.



Step 2 Click **Profile**. At the bottom of the profile details page, enter the device name and verification code. You can specify the device name by yourself. The verification code is the IMEI ID on the module or obtained using AT+CGSN=1. Click **Register** to complete the registration.



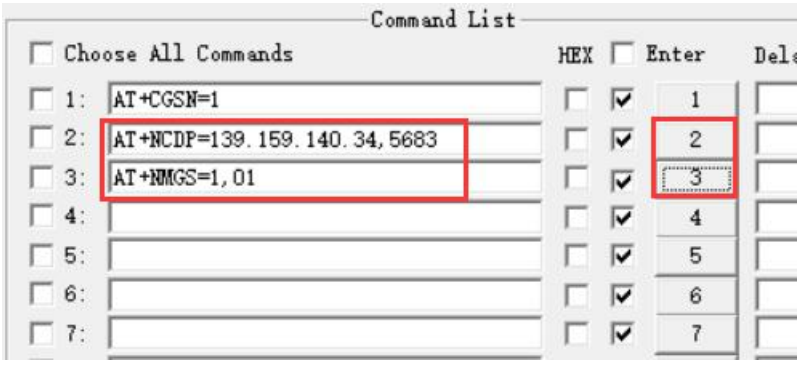
NOTE

You can click the modification or deletion button next to a device to modify or delete it.

----End

3.3.5 Reporting Data

Step 1 Start the serial port assistant and send the corresponding AT command, as shown in the following figure.

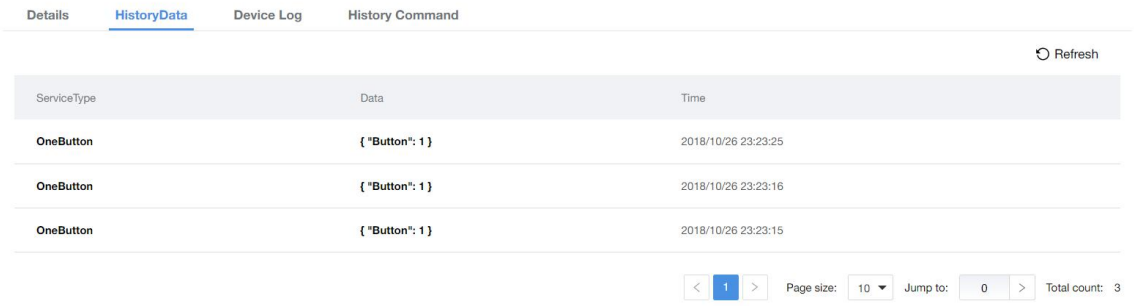


Step 2 In this example, one byte "01" is reported, indicating that the reported data corresponds to the **batteryLevel** field. The content is in one-to-one correspondence with the profile and the encoding and decoding plug-in.

NOTE

The IP address after AT+NCDP is subject to the device interconnection address in the email received after the platform is applied for.

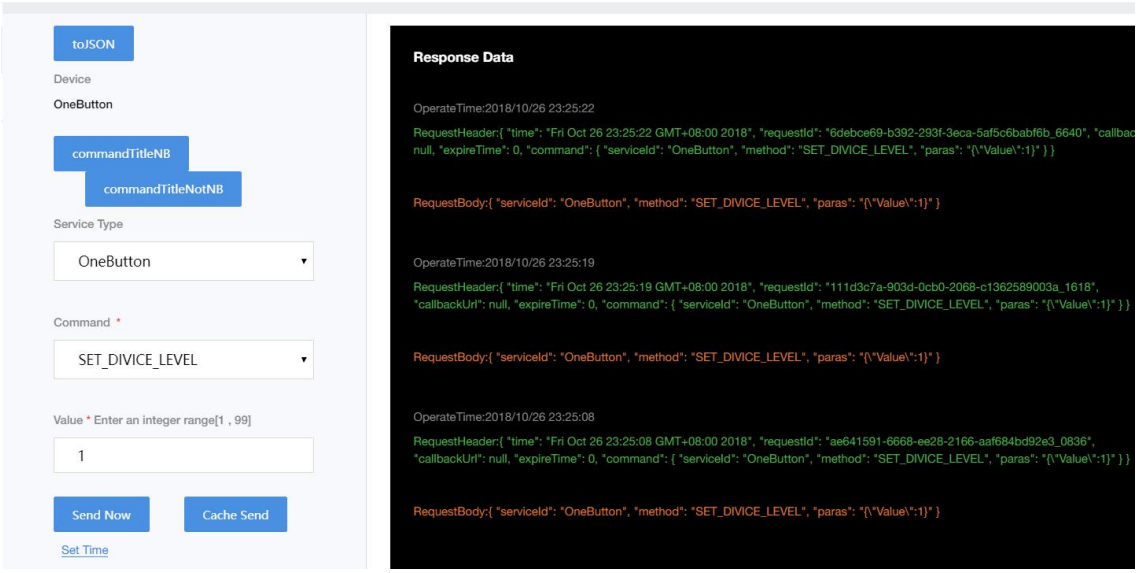
Step 3 Choose **My Devices** from the navigation tree. On the device details page that is displayed, click the **HistoryData** tab to view reported data. In this case, the data is reported successfully, as shown in the following figure.



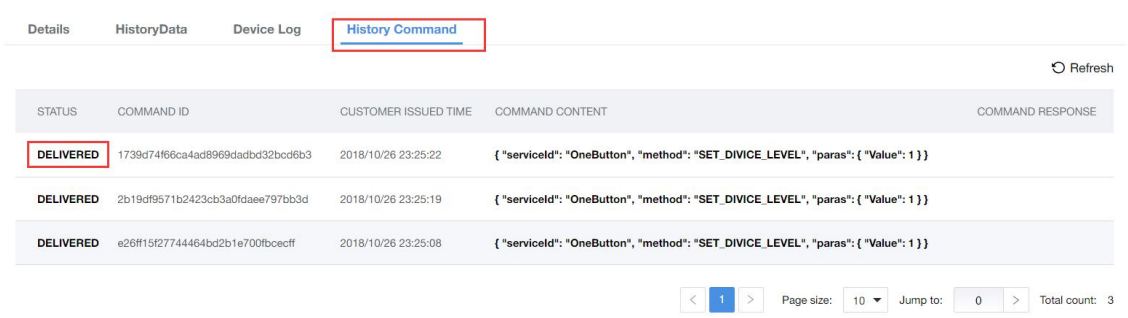
----End

3.3.6 Issuing a Command

Step 1 Choose **My Devices** from the navigation tree. On the device details page that is displayed, click the icon issuing icon </> on the right of the device. On the displayed command issuing page, select a command according to the displayed information, and then enter the values of command parameters, as shown in the following figure.

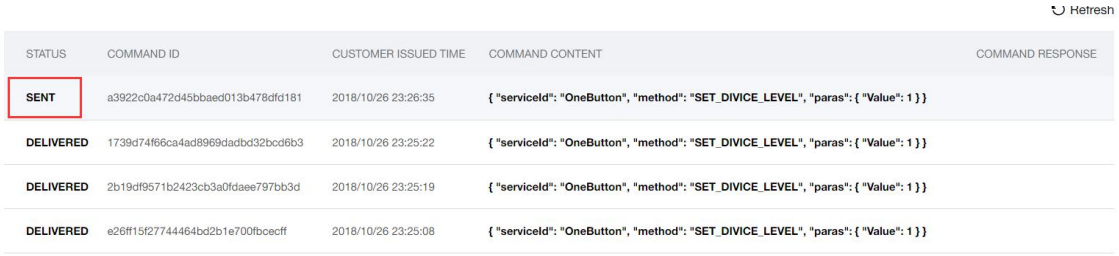


Step 2 Click **Send Now**. If the sending is successful, the details about the delivered data are displayed on the right pane of the page. In this case, click a device to view history commands. The message is in **DELIVERED** state, as shown in the following figure.

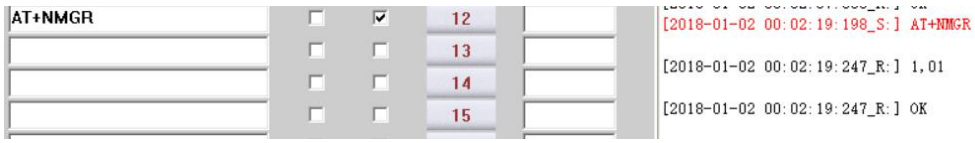


NOTE

If information in the following figure is displayed, the module enters the **Idle** or **PSM** state. In this case, the module enters the **Connected** state after the data is reported again, and the command can be delivered to the module. If the time is too long, the command delivery times out.



Step 3 Send AT+NMGR using the serial port to view the delivered data, as shown in the following figure.



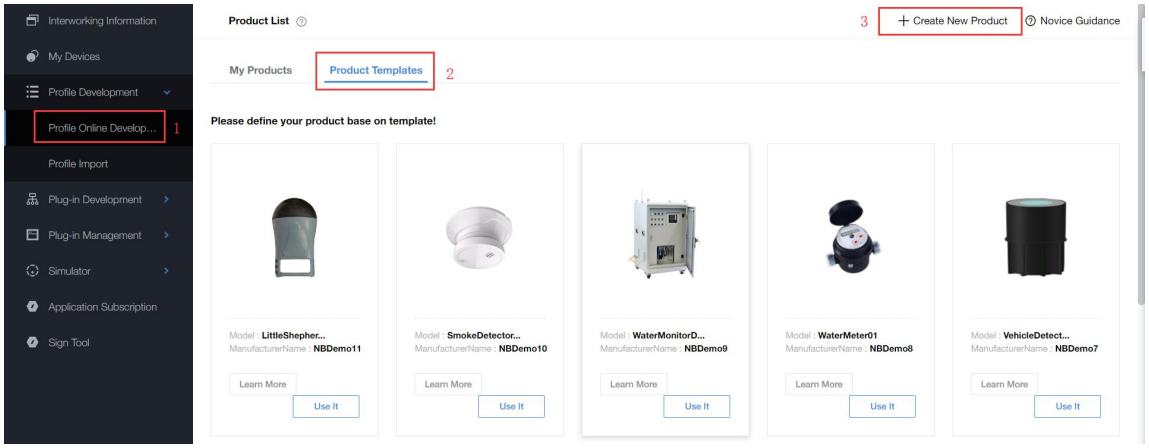
----End

3.4 Practice of Developing the Smart Street Lamp Using the NB-IoT

3.4.1 Creating a Profile

3.4.1.1 Creating a Product

Note that the device model, manufacturer ID, and manufacturer name must be unique.



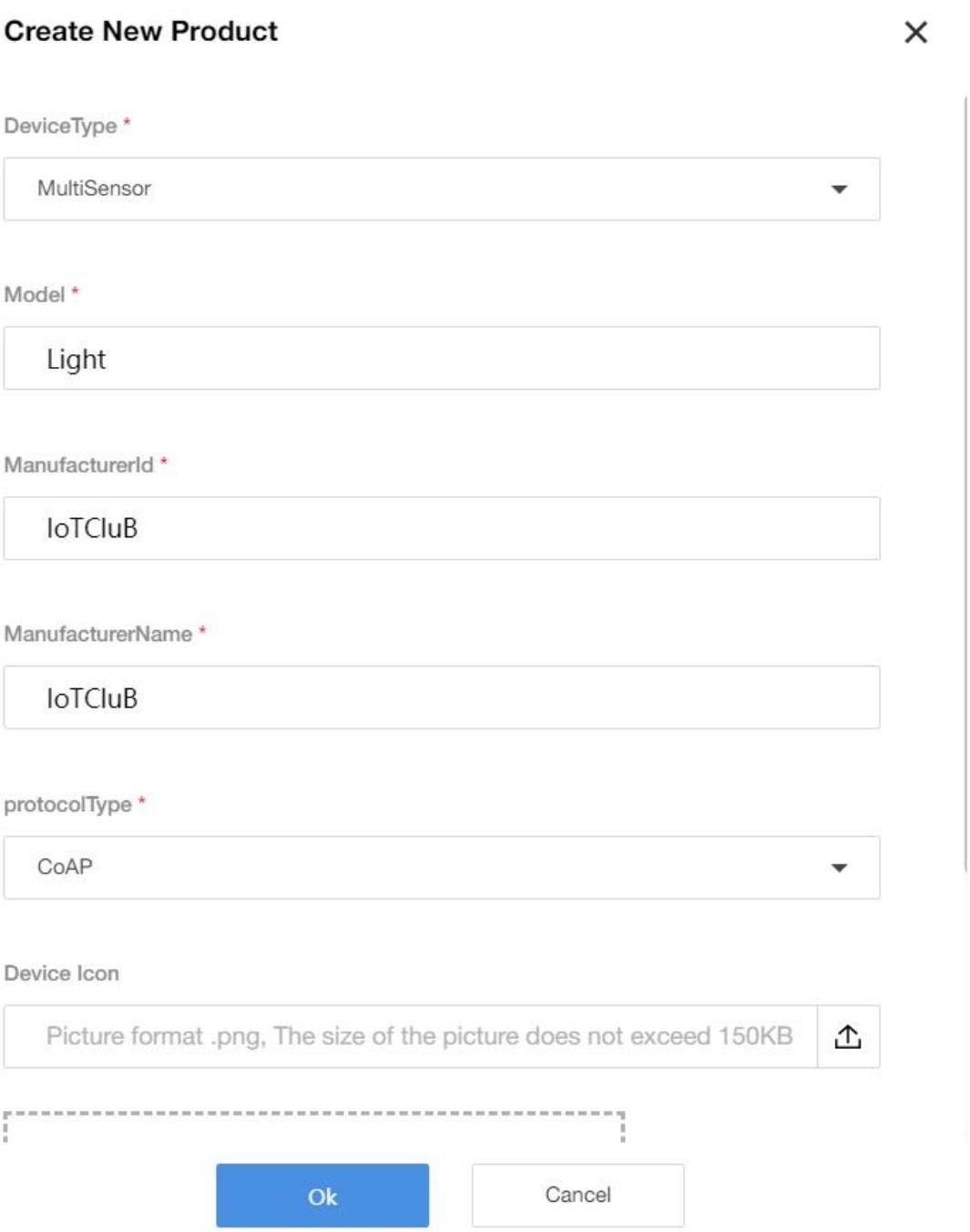
Device type: MultiSensor

Device model: Light

Manufacturer ID: any

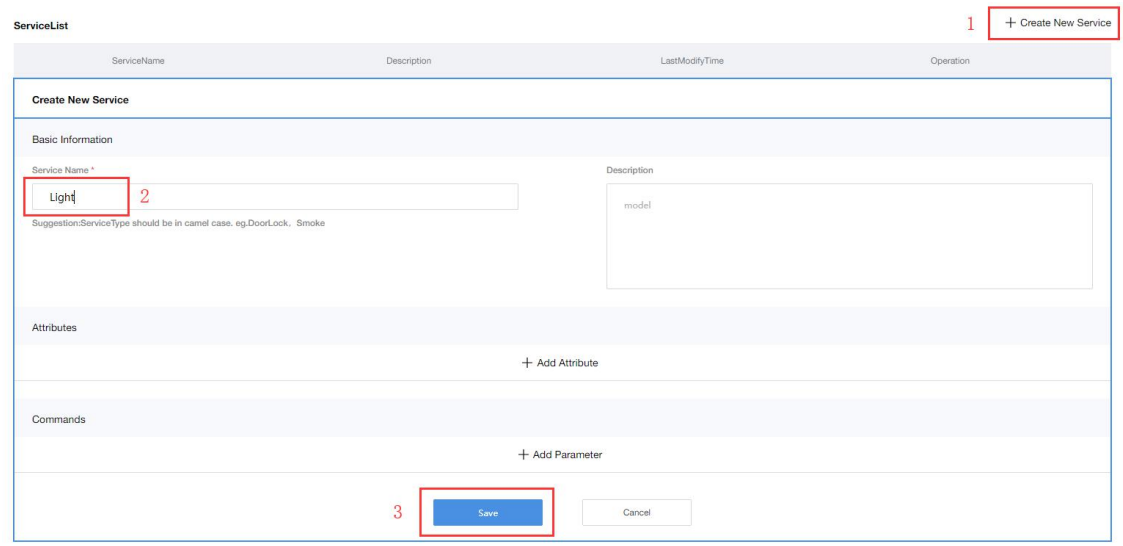
Manufacturer name: same as the manufacturer ID

Protocol type: CoAP

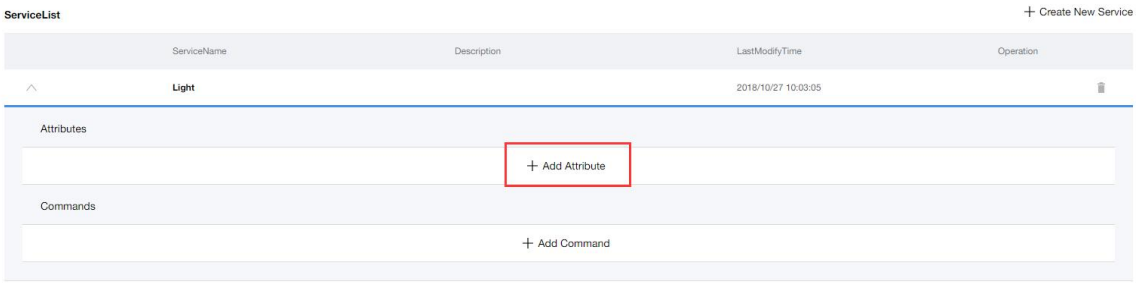


3.4.1.2 Creating a Service

Service name: Light



3.4.1.3 Adding an Attribute

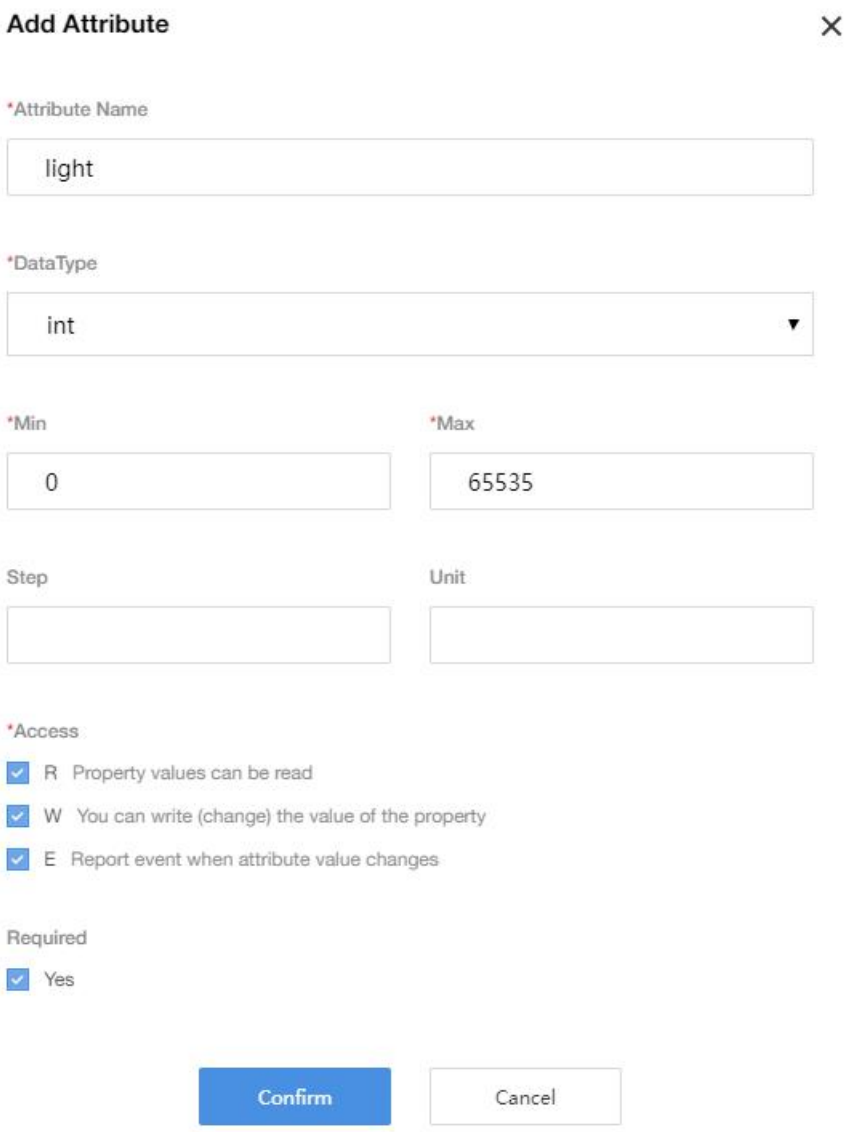


Attribute name: light

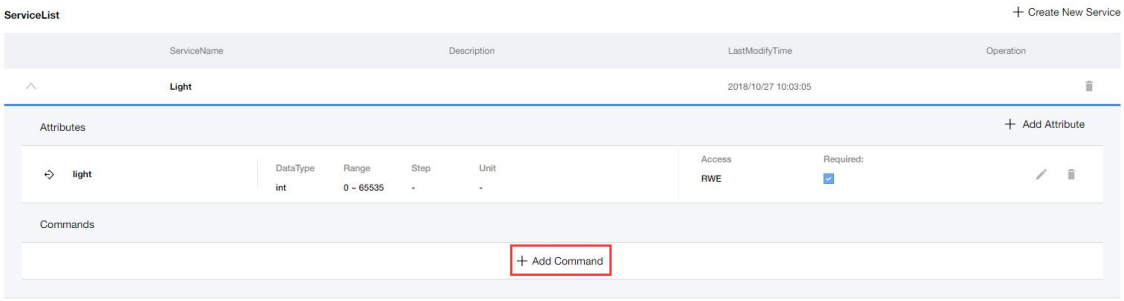
Attribute type: int

Minimum value: 0

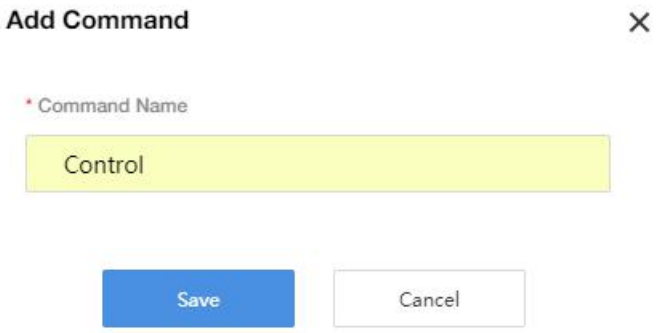
Maximum value: 65535



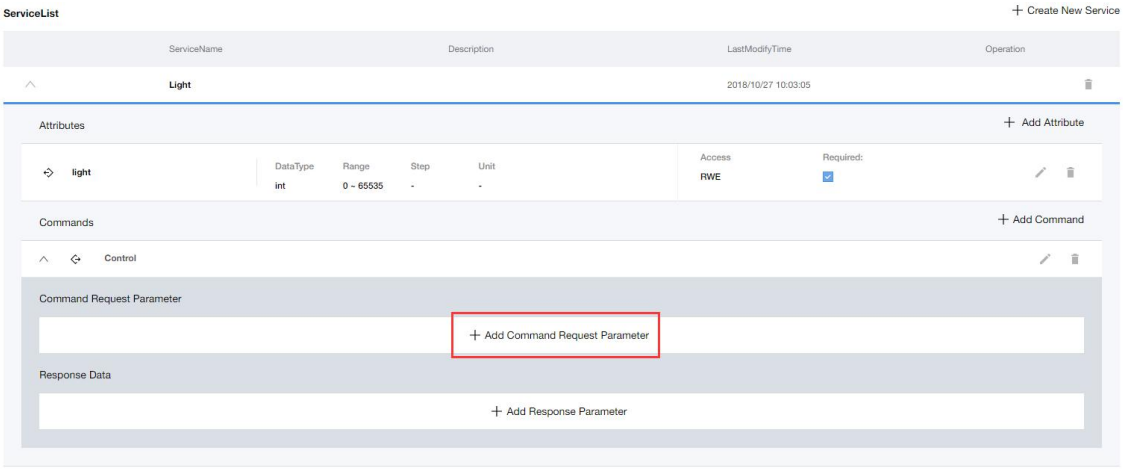
3.4.1.4 Adding a Command



Command name: Control



3.4.1.5 Adding a Command Issuing Parameter

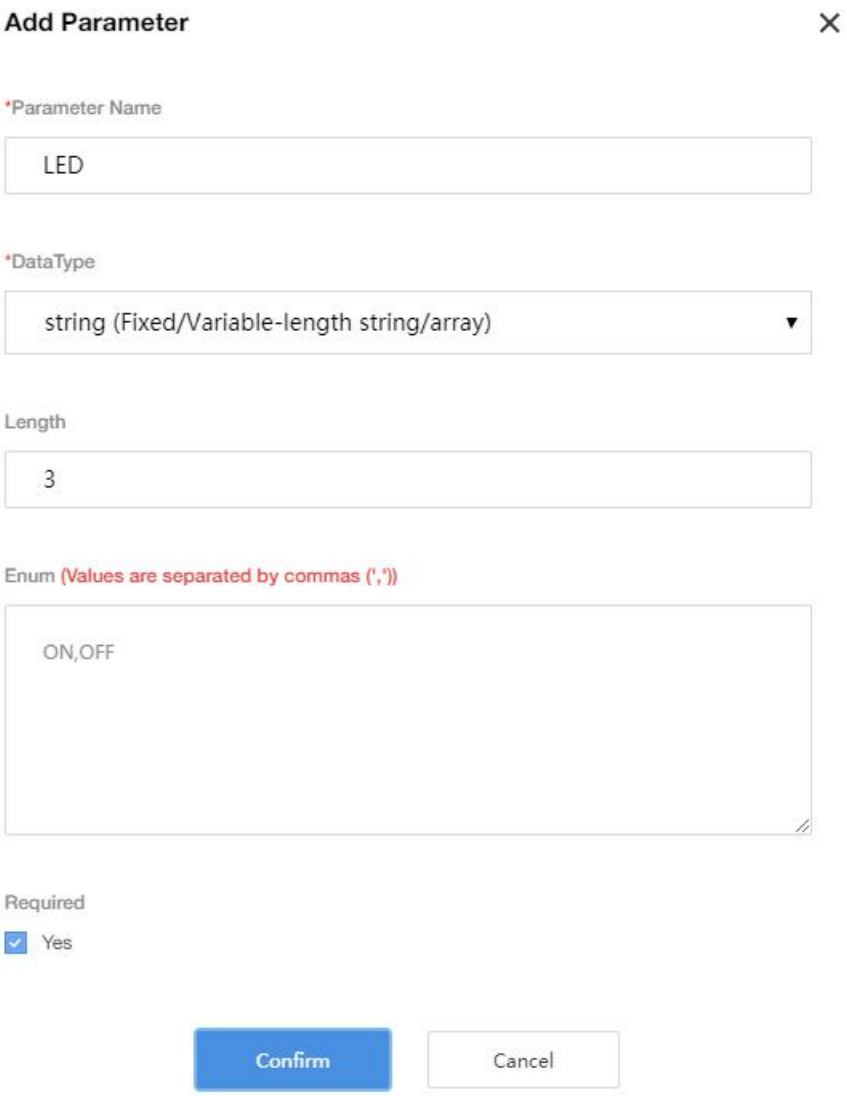


Parameter name: LED

Attribute type: string

Length: 3

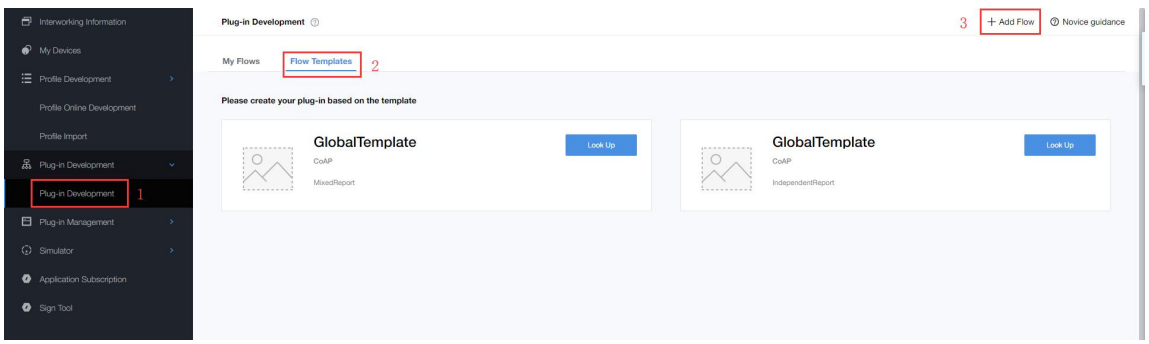
Enumerated values: ON and OFF

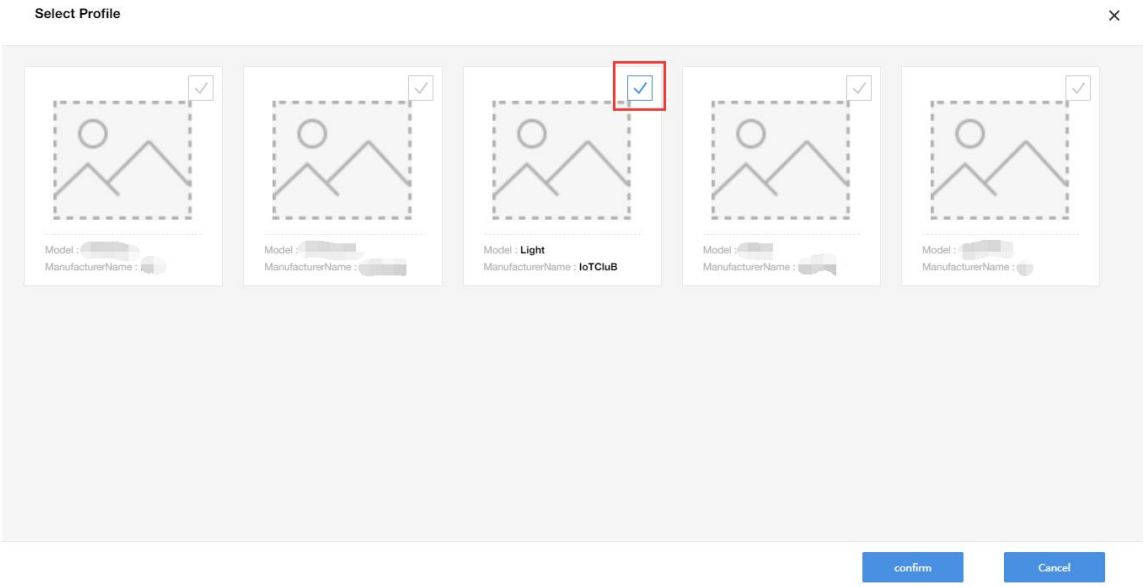


The profile creation is complete.

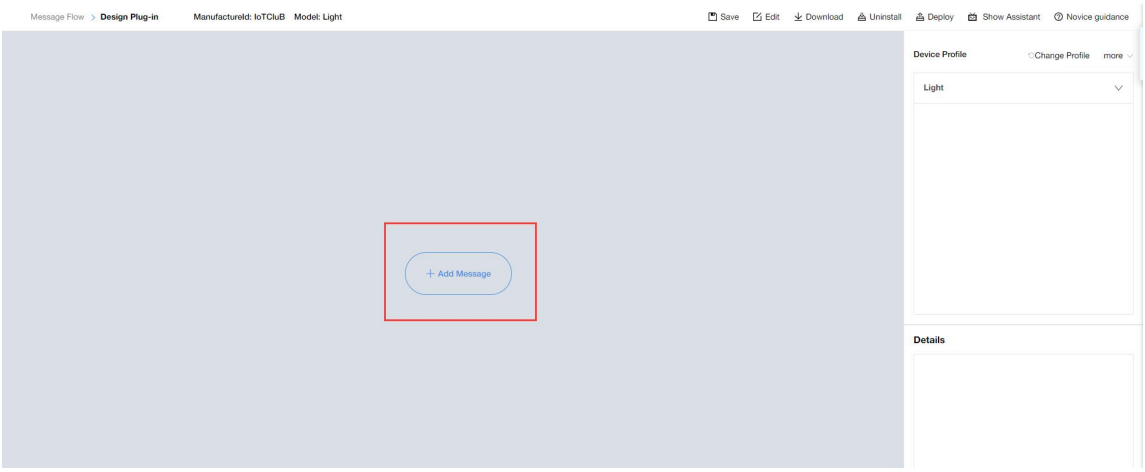
3.4.2 Developing the Encoding and Decoding Plug-in

3.4.2.1 Developing the Plug-in



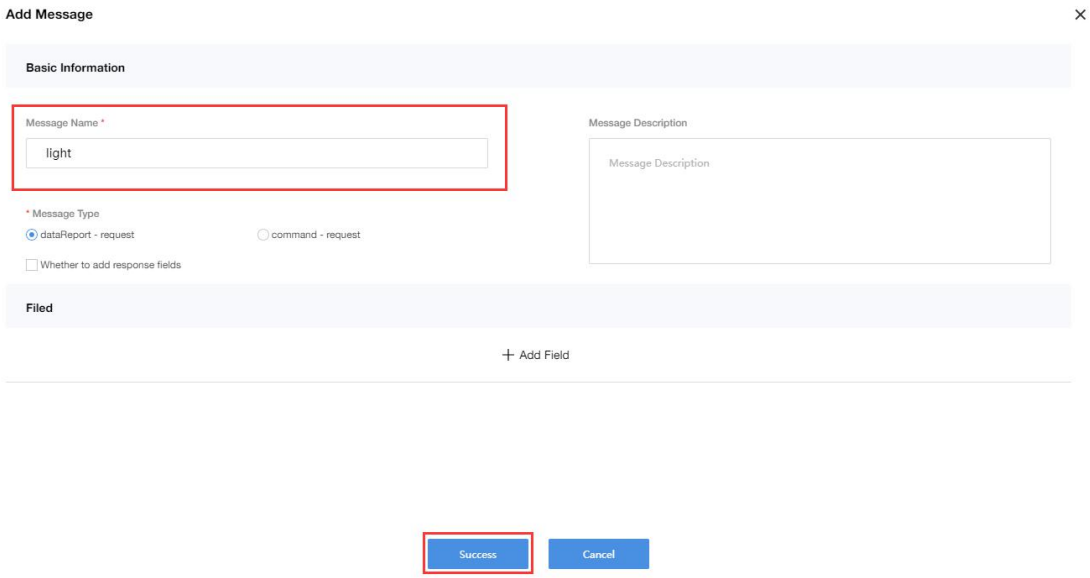


3.4.2.2 Adding a Data Reporting Message

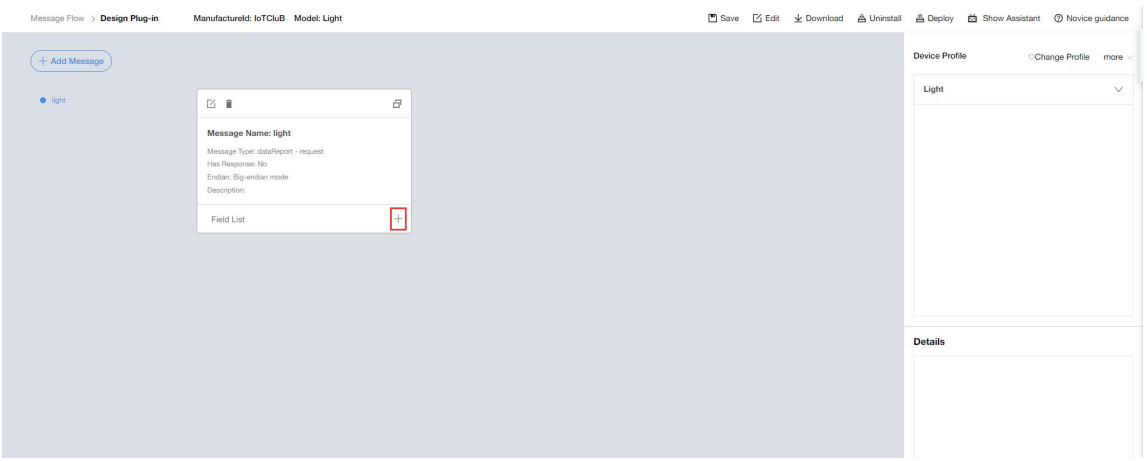


Message name: light

Message type: data reporting



3.4.2.3 Adding a Data Reporting Parameter



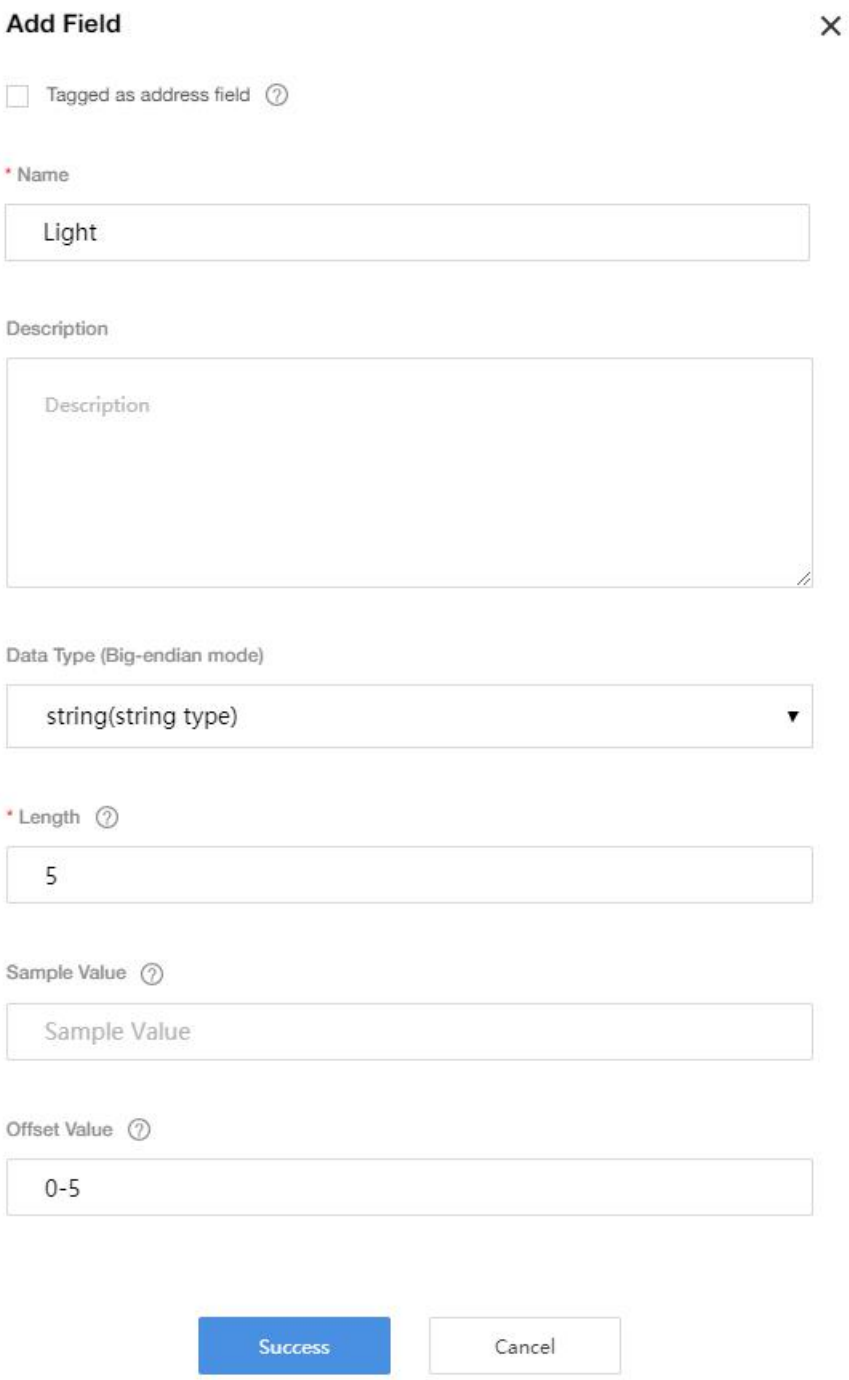
Name: Light

Description: This field is optional.

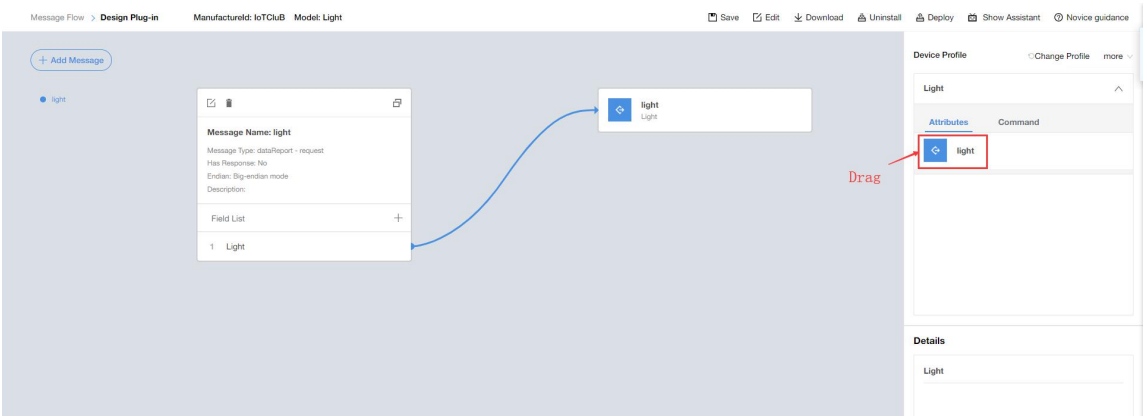
Data type: string

Length: 5

Default value: This field is optional.


Add Field
Tagged as address field
* Name
Light
Description
Description
Data Type (Big-endian mode)
string(string type)
* Length
5
Sample Value
Sample Value
Offset Value
0-5
Success
Cancel

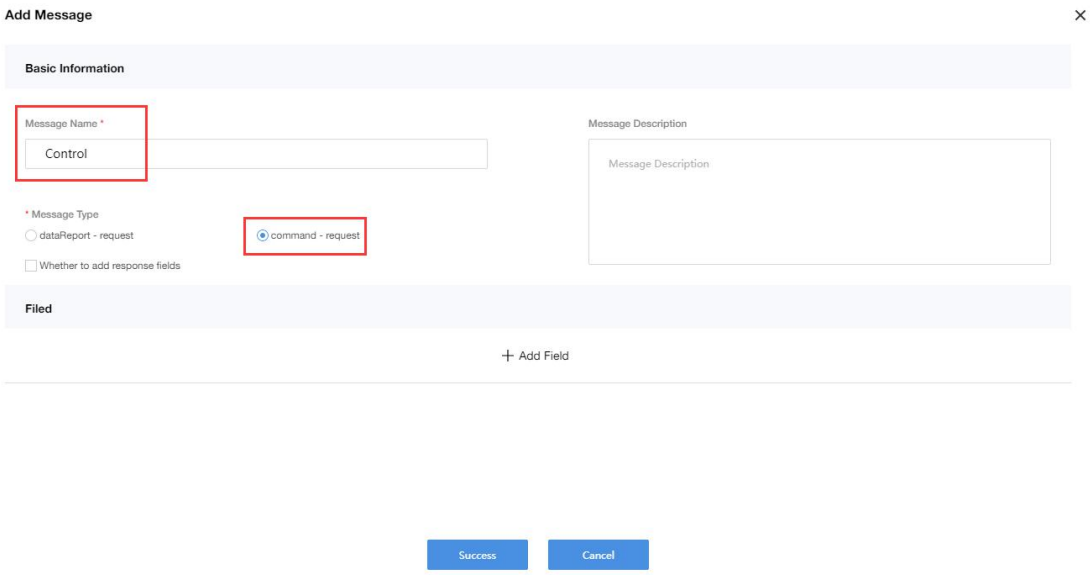
3.4.2.4 Creating a Data Reporting Mapping Profile



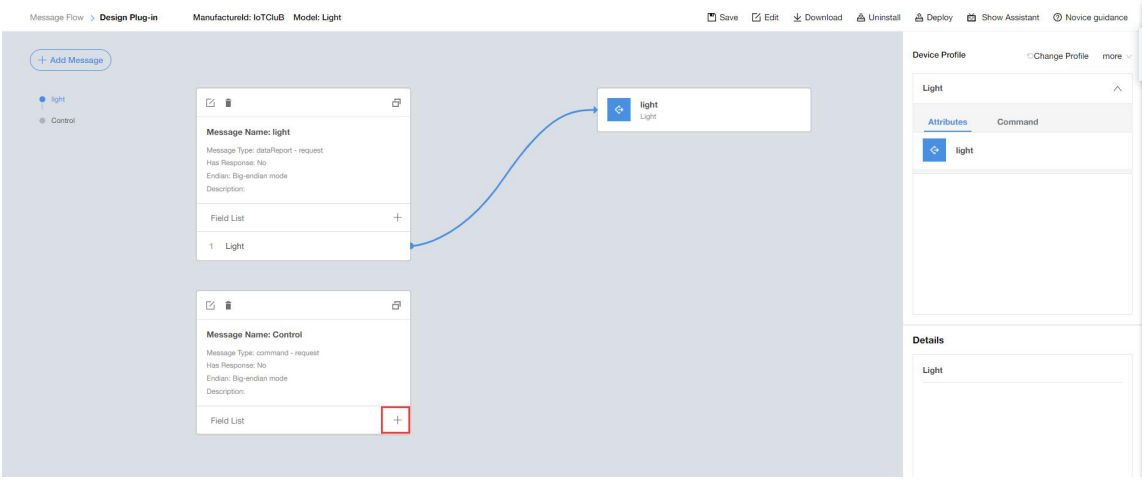
3.4.2.5 Adding a Command Issuing Message

Message name: Control

Message type: command issuing



3.4.2.6 Adding a Data Reporting Parameter



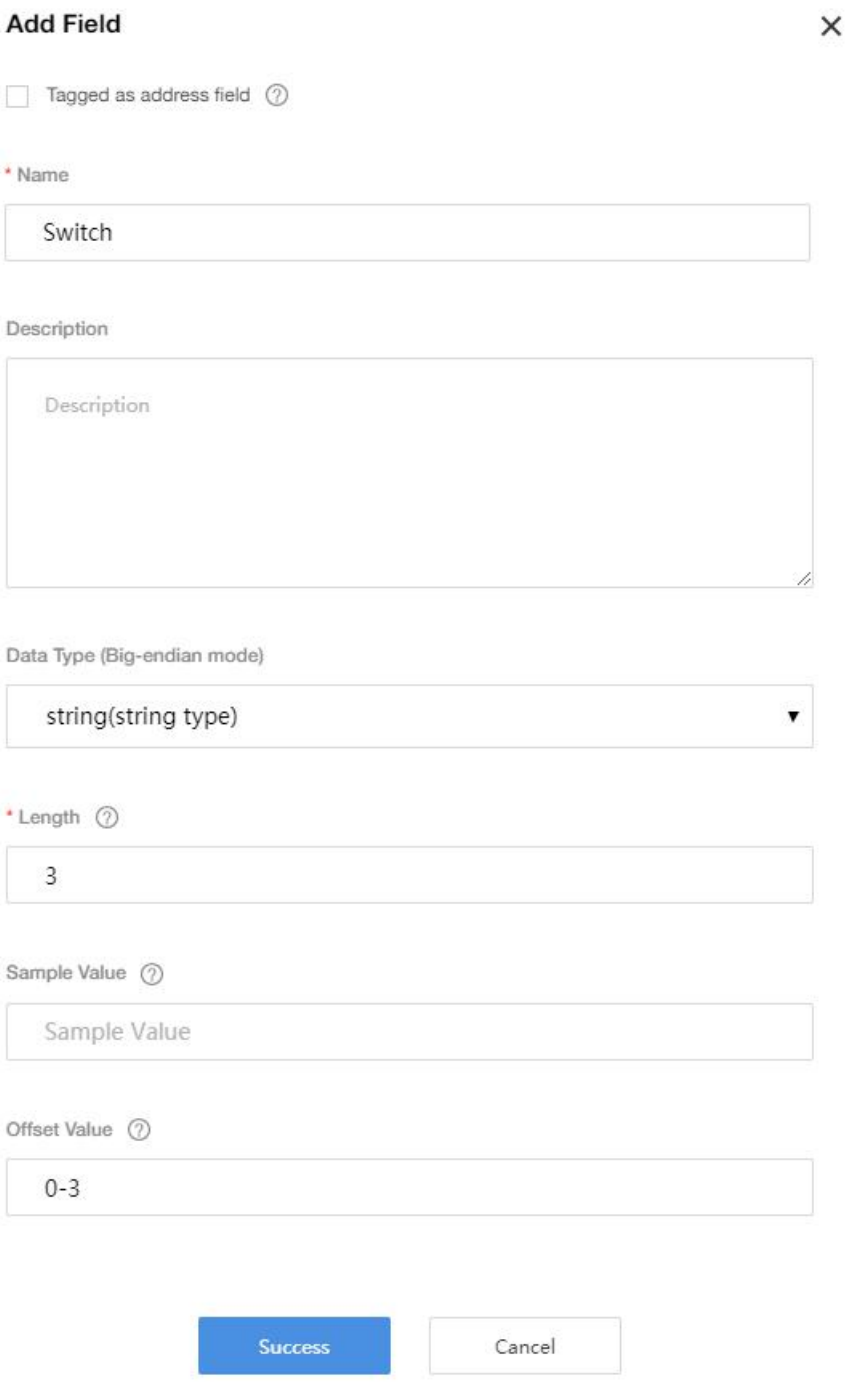
Name: Switch

Description: This field is optional.

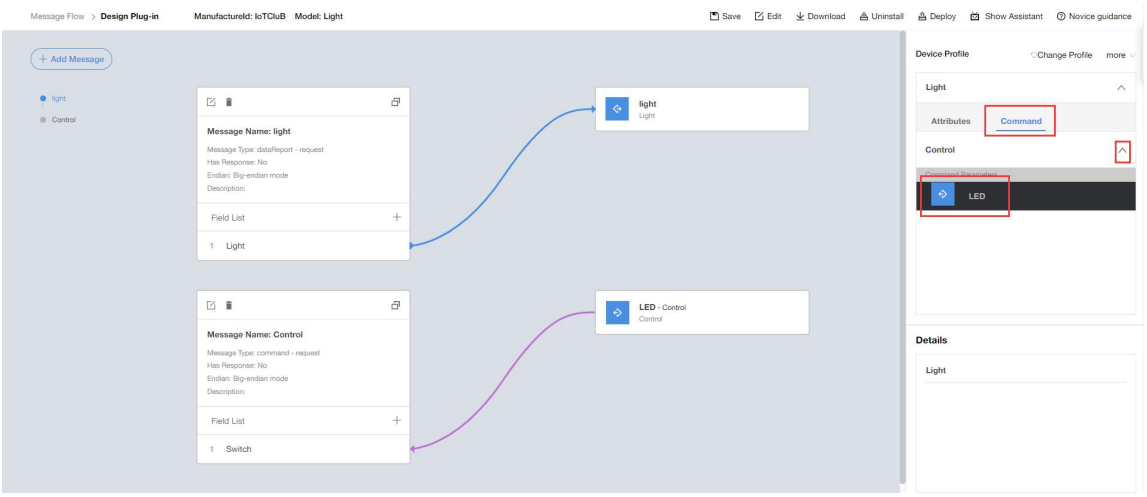
Data type: string

Length: 3

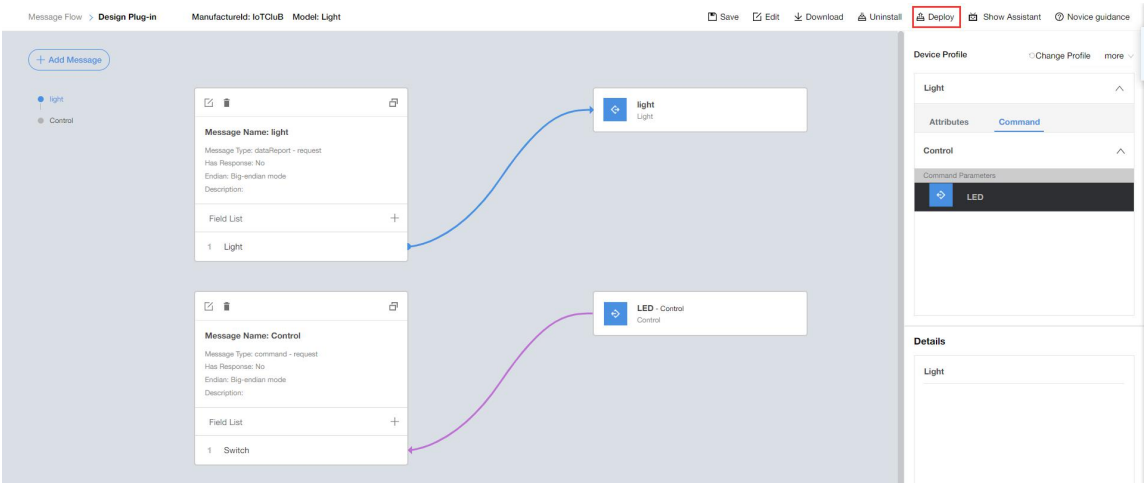
Default value: This field is optional.

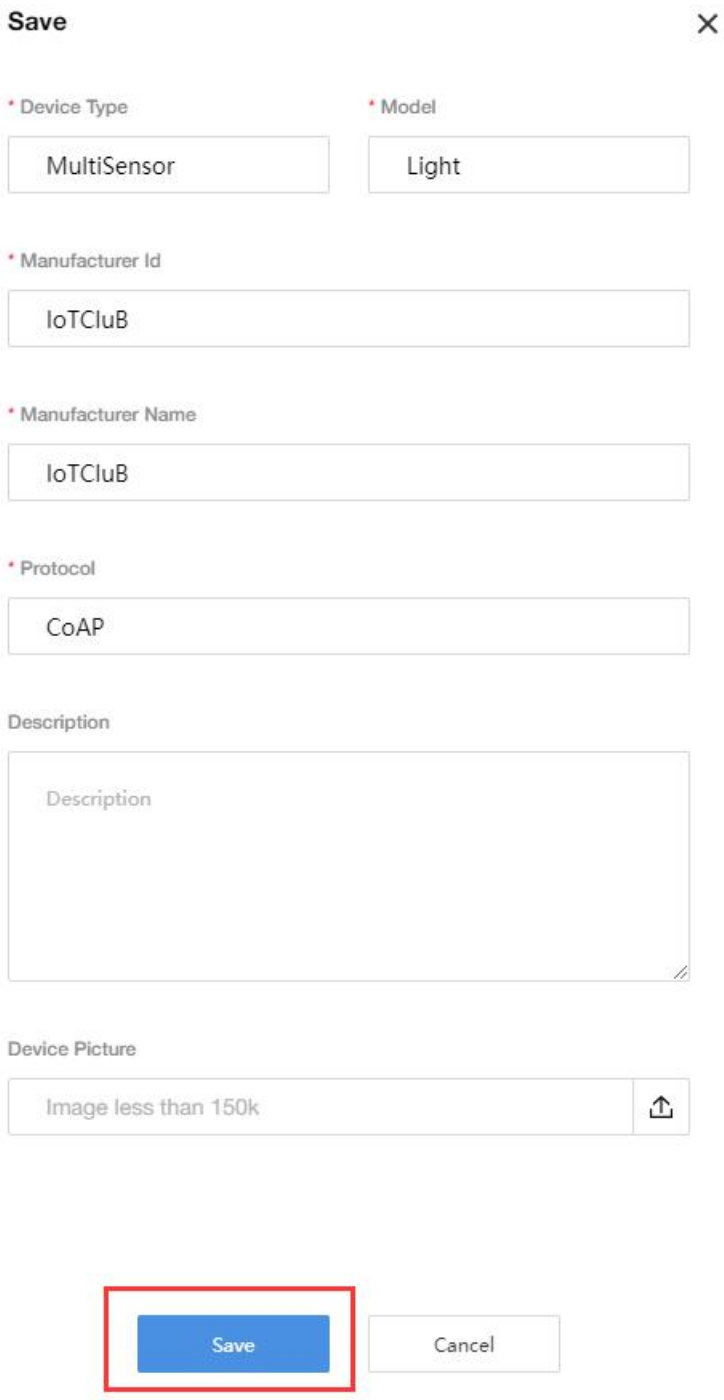

Add Field
Tagged as address field
* Name
Switch
Description
Description
Data Type (Big-endian mode)
string(string type)
* Length
3
Sample Value
Sample Value
Offset Value
0-3
Success
Cancel

3.4.2.7 Creating a Command Issuing Mapping Profile



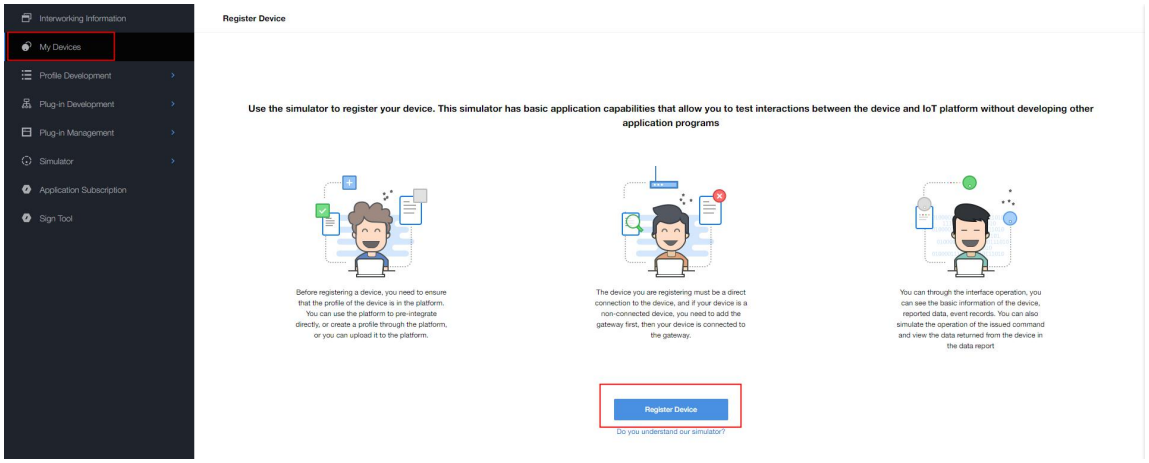
3.4.2.8 Deploying the Plug-in

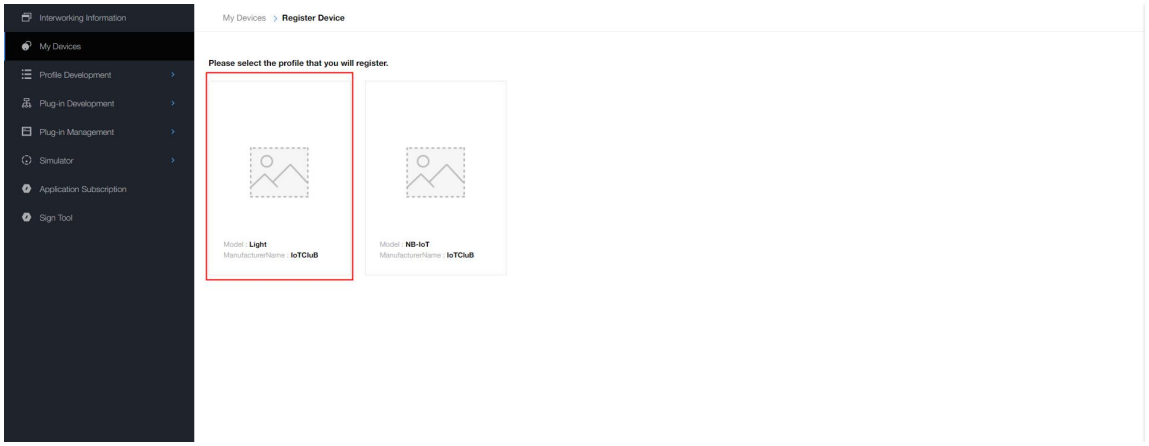


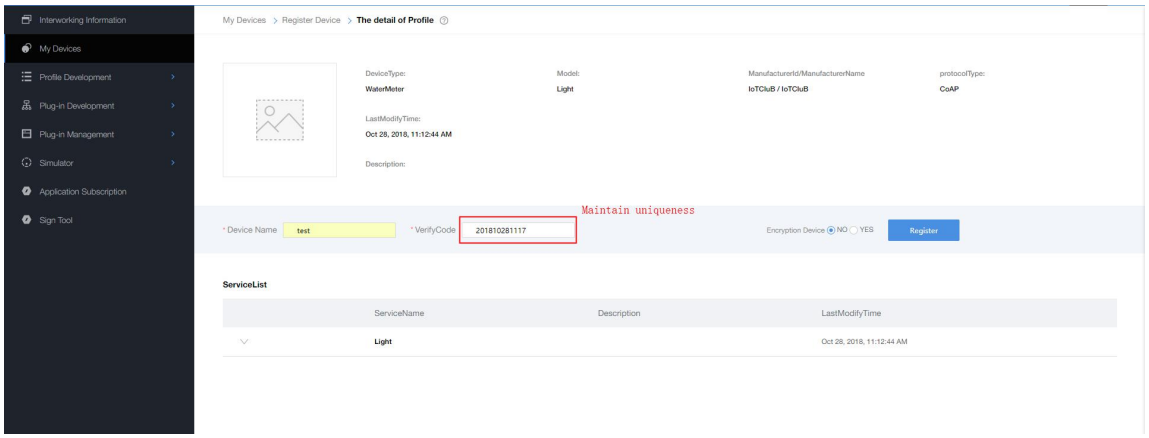

Save
* Device Type
MultiSensor
* Model
Light
* Manufacturer Id
IoTCluB
* Manufacturer Name
IoTCluB
* Protocol
CoAP
Description
Description
Device Picture
Image less than 150k
Save
Cancel

3.4.3 Online Simulator

3.4.3.1 Registering a Simulator

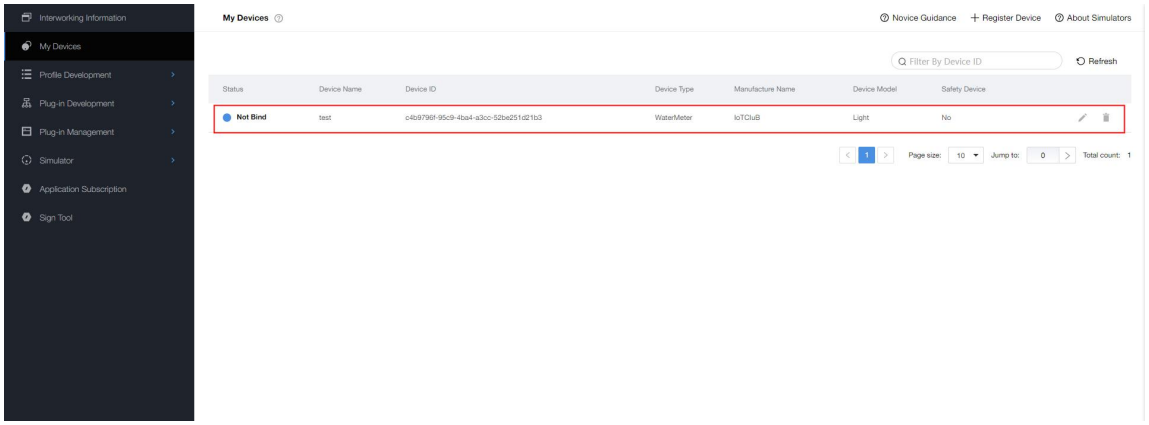


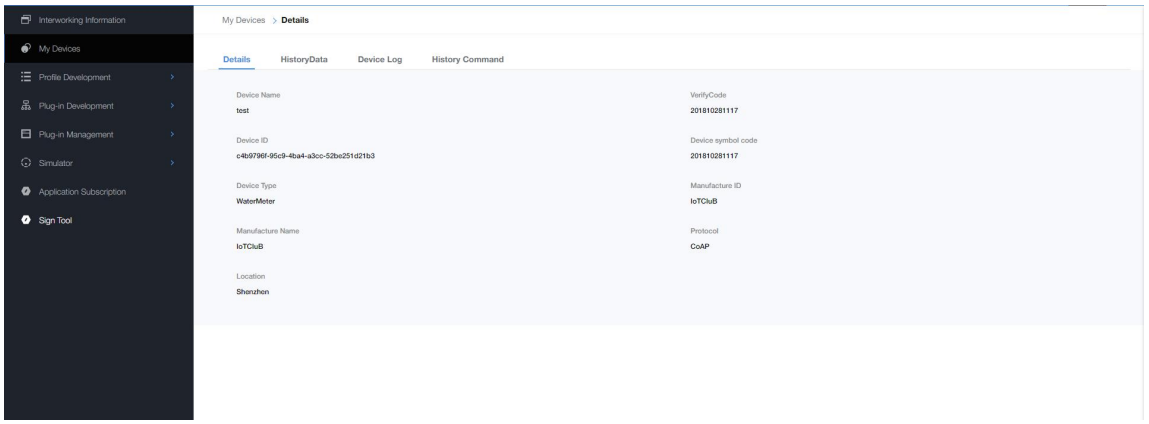




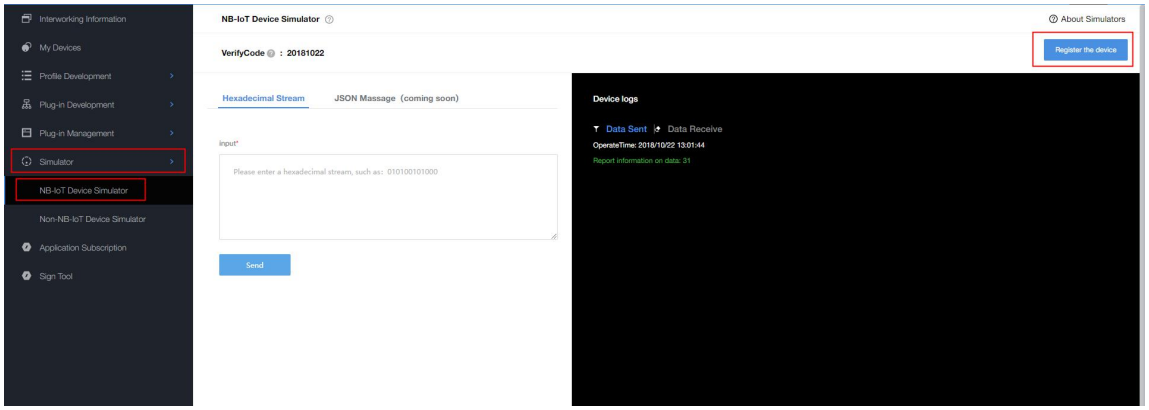
Note：The **VerifyCode** here should be unique, and please use the **IMEI** number as **VerifyCode** for real device.

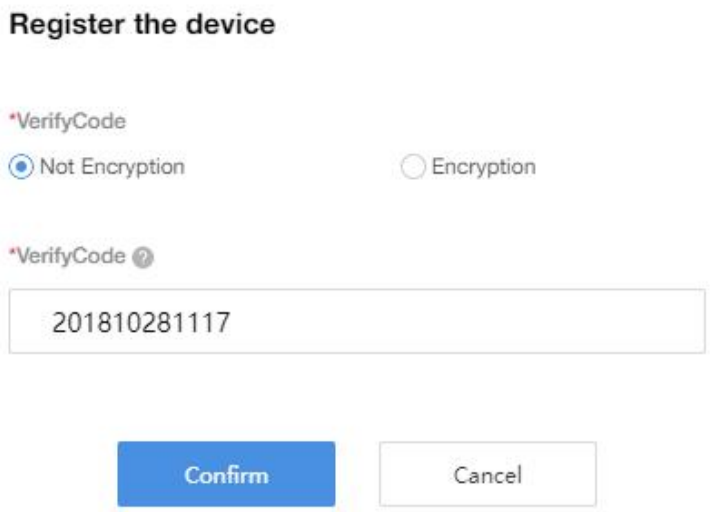
3.4.3.2 Viewing Simulator Device Details





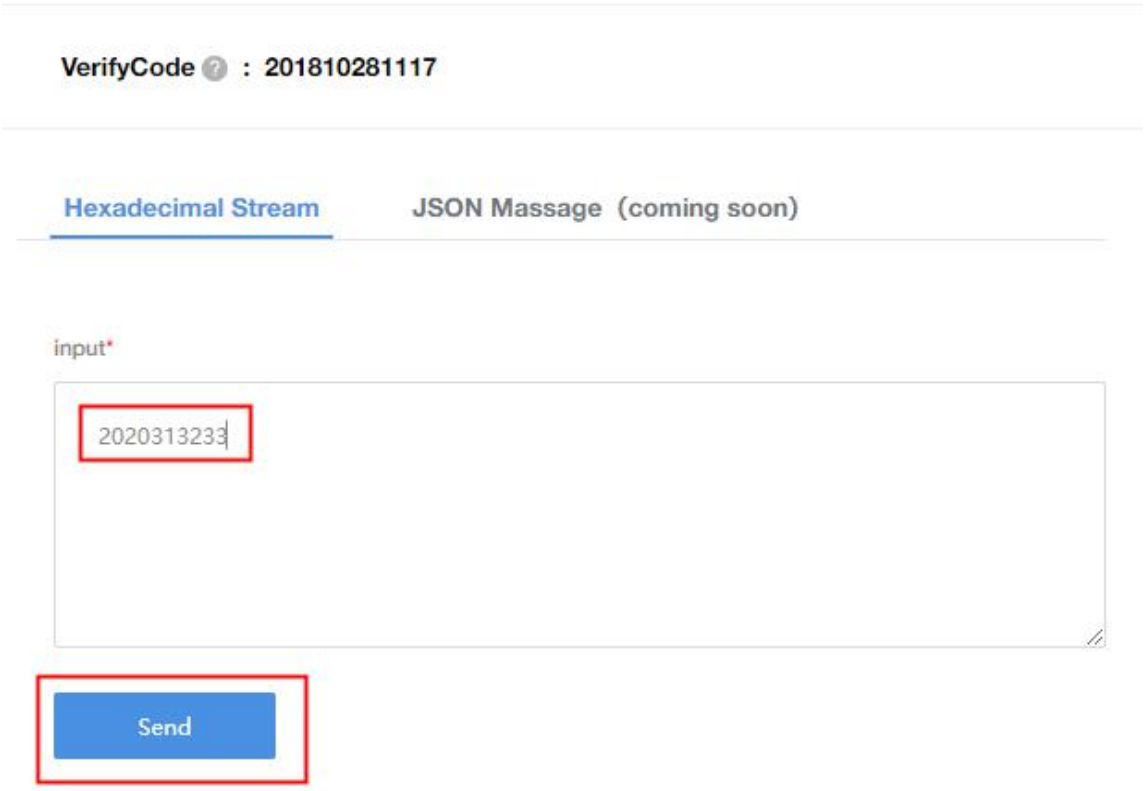
3.4.3.3 Binding the Simulator to a Device



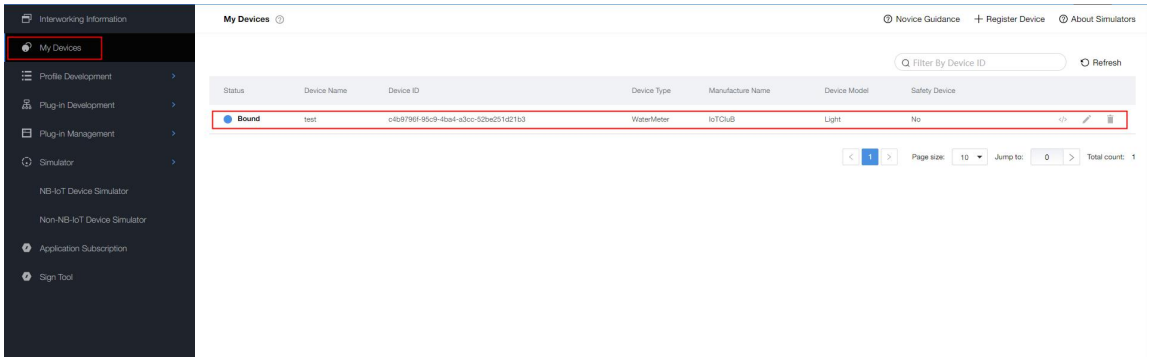


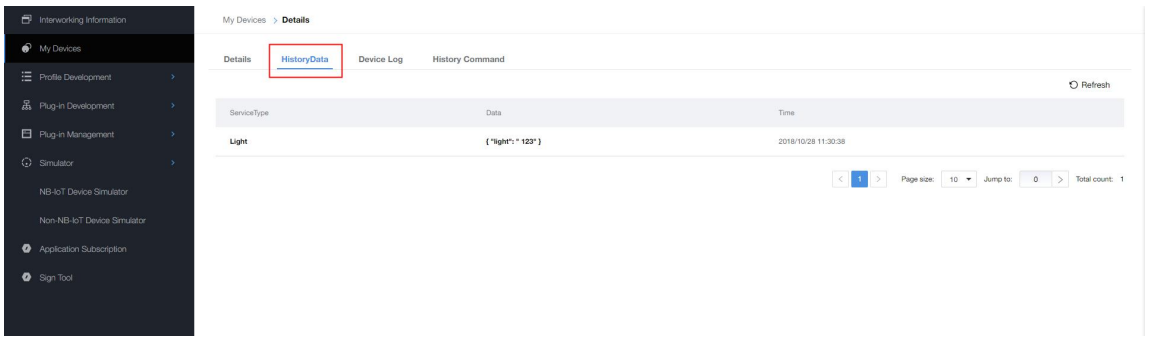
The verification code is consistent with the previously registered one.

3.4.3.4 Sending Data

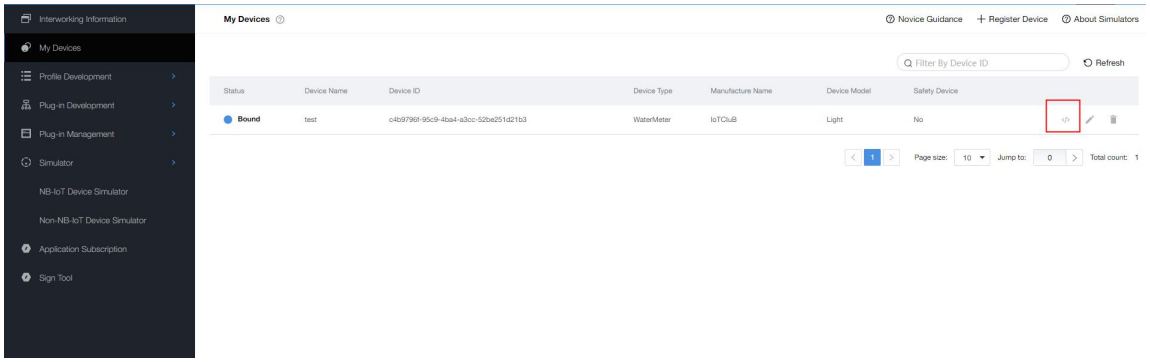


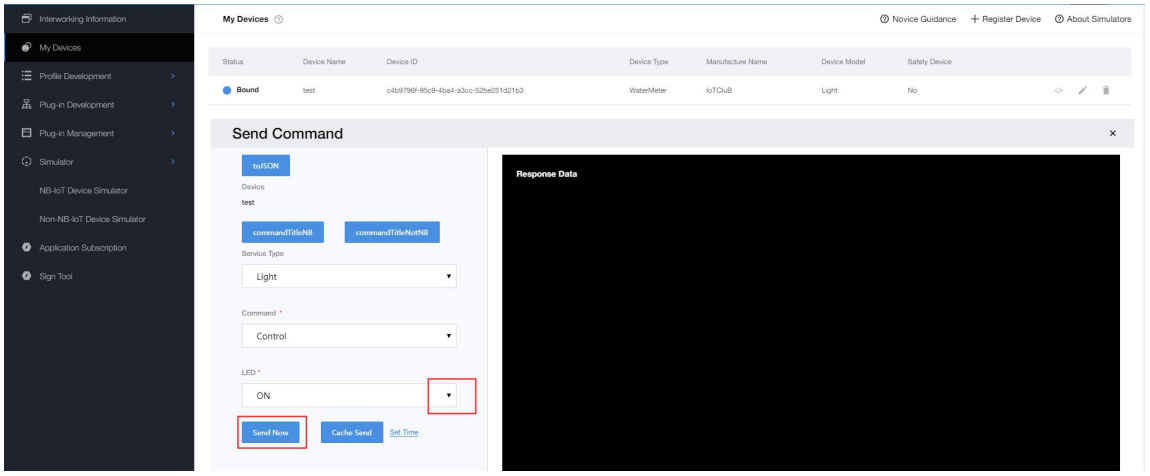
3.4.3.5 Viewing Historical Data of the Simulator



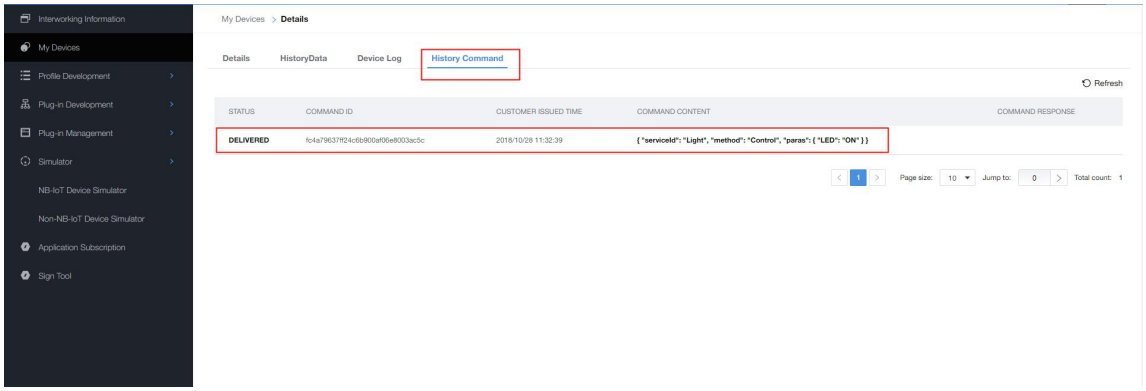


3.4.3.6 Issuing a Command





3.4.3.7 Viewing Historical Commands of the Simulator



The verification on the creation of the profile and encoding and decoding plug-in is complete.

3.4.4 Connecting the NB-IoT Module to the Platform

3.4.4.1 NB-IoT: Parsing AT Commands

NOTE

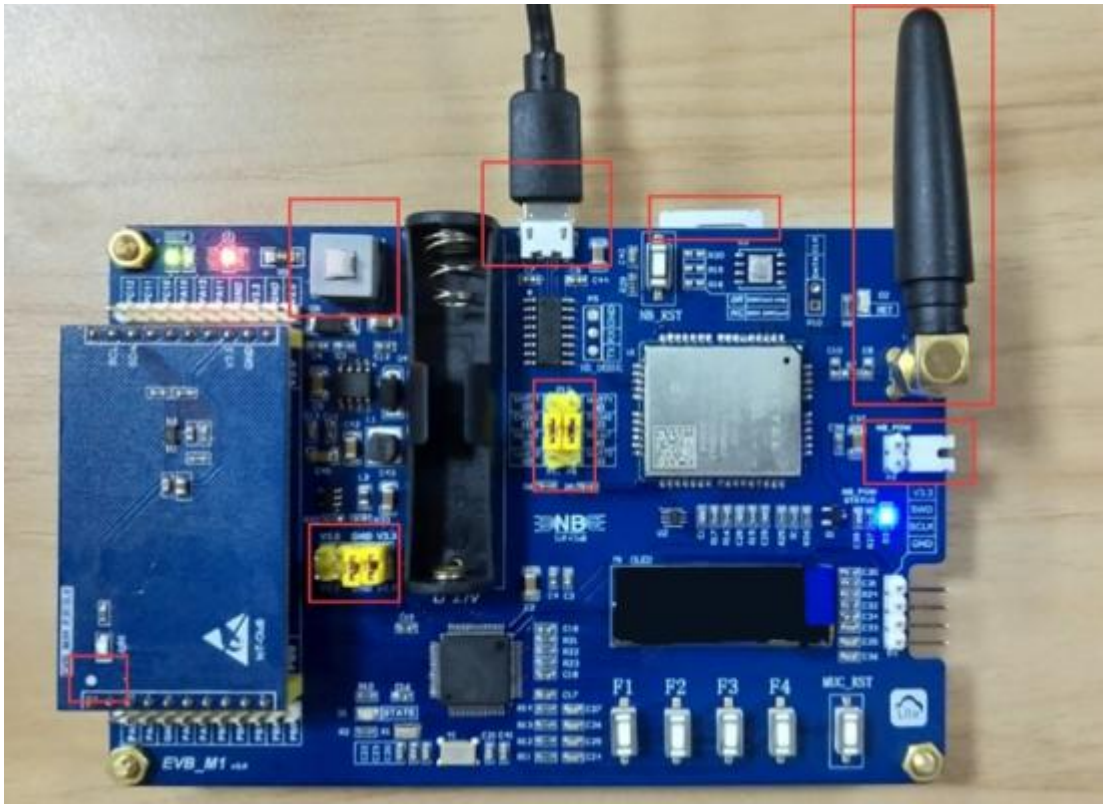
For details about the AT commands, see the NB-IoT module command set manual.

Command	Description
AT+NRB	Reboots the module.
AT+CGMR	Queries the manufacturer and version information.
AT+NBAND?	Queries the frequency band. 5 indicates 850 MHz of China Telecom. 8 indicates 900 MHz of China Mobile and China Unicom.
AT+CGSN=1	Queries the product SN. If the value is **1**, the IMEI is returned.
AT+NCDP=218.4.33.72,5683	Configures CDP server parameters (IP address and port number).
AT+NCDP?	Queries CDP server settings.
AT+CEREG=1	Queries the network registration status. **1** indicates that the auto reply of network registration is enabled. **0** indicates that the auto reply of network registration is disabled.
AT+CMEE=1	Indicates the type of the mobile terminal error reported. **1** indicates that the detailed error code is returned. **0** indicates that ERROR is returned for all types of errors.
AT+NSMI=1	Indicates whether the message TX result is displayed 0: no 1: yes
AT+NNMI=1	Indicates whether the flag of a new message is displayed. 0: no 1: yes. The flag and data are displayed.
AT+CIMI	Queries the IMEI.

AT+CSQ	Obtains the signal strength:
AT+NUESTATS	Obtains the module status (including SNR and CELLID).
AT+CGATT?	Indicates whether the module is attached to a base station. 0: no 1: yes
AT+NMSTATUS?	Displays the information registration status.
AT+NMGS=5,2020373839	Indicates the sent message.
AT+NMGR	Indicates the received message.
AT+NQMGS	Displays the statistics on sent messages.
AT+NQMGR	Displays the statistics on received messages.

3.4.4.2 Manually Operating the NB-IoT

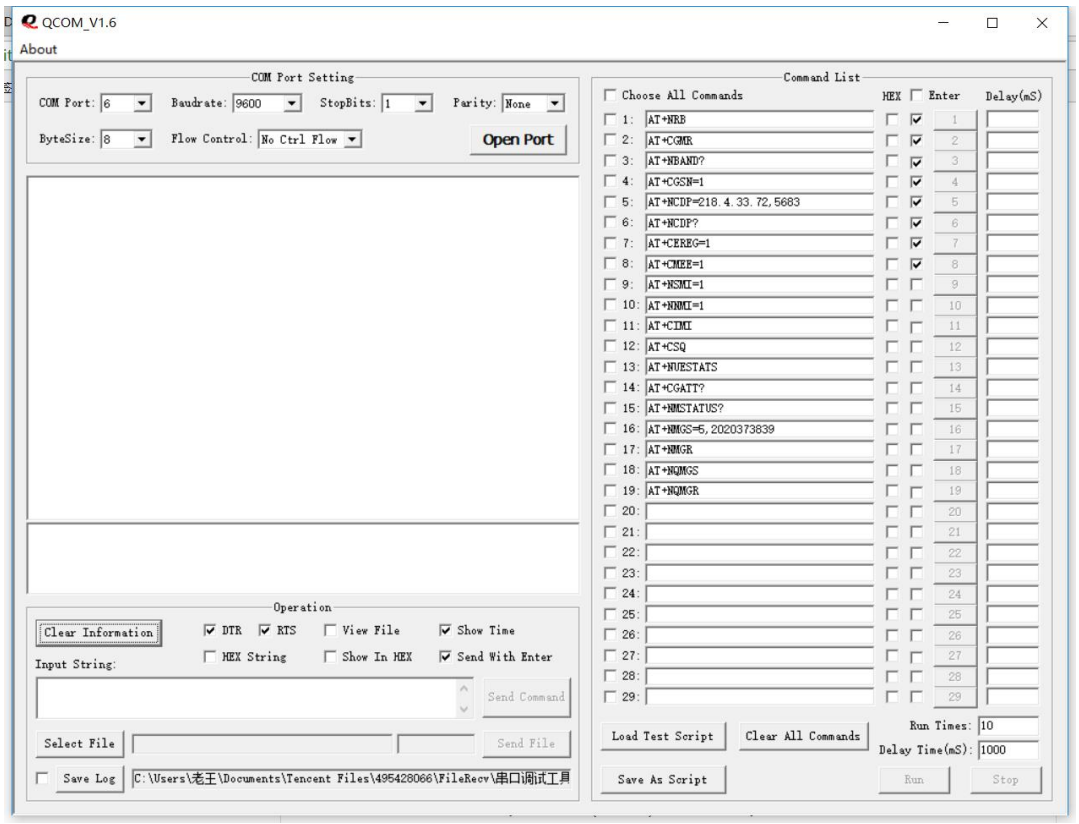
Use the serial port debugging software to manually send AT commands to the NB-IoT module. The information returned by the NB-IoT module is directly connected to the serial port of the computer for debugging. The operations are performed manually step by step to connect the NB-IoT module to the OceanConnect platform. The CoAP protocol is used for communication.

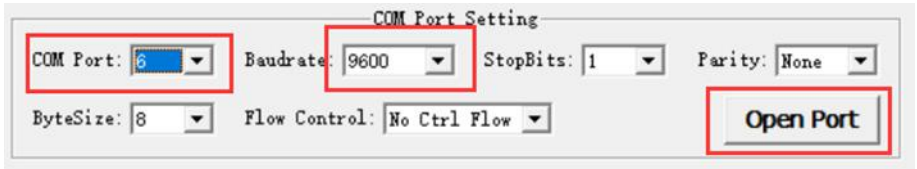


3.4.4.3 Connecting the Hardware Cables of the Development Board

Note the cable connections of the P12 jumper cap and burner.

3.4.4.4 Communication Between the NB-IoT and Serial Port Debugging Assistant





- Baud rate: 9600. Select the serial port number according to the actual situation, and then click **Open Port**.
- Command 1: **AT+NRB**

  REBOOTING

  0???

  REBOOT_CAUSE_APPLICATION_AT

  Neul

  OK
- Command 2: **AT+CGMR**. The latest version of the module is **657SP5**, which may be upgraded later. If the version of the used module is earlier than 657SP5, upgrade the module by referring to the module upgrade tutorial.

  SECURITY,V100R100C10B657SP5

  PROTOCOL,V100R100C10B657SP5

  APPLICATION,V100R100C10B657SP5

  SEC_UPDATER,V100R100C10B657SP5

  APP_UPDATER,V100R100C10B657SP5

  RADIO,BC95HB-02-STD_850

  OK
- Command 3: **AT+NBAND?**

  +NBAND:5

  OK
- Command 4: **AT+CGSN=1**. For modules with the version of 657SP2 or later, you can run this command directly to read the IMEI.

  +CGSN:863703330633304

  OK
- Command 5: **AT+NCDP=218.4.33.72, 5683**. If a Huawei cloud platform other than OceanConnect is used, change the IP address and port number based on the actual situation.

  OK
- Command 6: **AT+NCDP?**

  +NCDP:218.4.33.72,5683

  OK
- Command 7: **AT+CEREG=1**

  OK

Command 8: **AT+CMEE=1**. The function of detailed error code is enabled. If this function is disabled, **ERROR** is returned for all types of occurs.

OK

- Command 9: **AT+NSMI=1**

  OK

- Command 10: **AT+NNMI=1**

  OK

- Command 11: **AT+CIMI**

  460111176313522

  OK

- Command 12: **AT+CSQ**

  +CSQ:25,99

  OK

- Command 13: **AT+NUESTATS**

  Signal power:-718

  Total power:-625

  TX power:-30

  TX time:652

  RX time:22771

  Cell ID:67208020

  ECL:1

  SNR:38

  EARFCN:2507

  PCI:301

  RSRQ:-125

  OK

- Command 14: **AT+CGATT?**

  +CGATT:1

  OK

- Command 15: **AT+NMSTATUS?**. This command is used to test the connection between the NB-IoT module and the IoT platform. If the message "+CME ERROR: 513" is returned after the AT+NMGS command is executed for the first time to report data, run the **AT+NMSTATUS?** Command until the module returns "+NMSTATUS:MO_DATA_ENABLED", indicating that the NB-IoT module has been connected to the IoT platform.

  +NMSTATUS:MO_DATA_ENABLED

  OK

- Command 16: **AT+NMGS=5,2020373839**. Data **789** is reported.

  OK

+NSMI:SENT

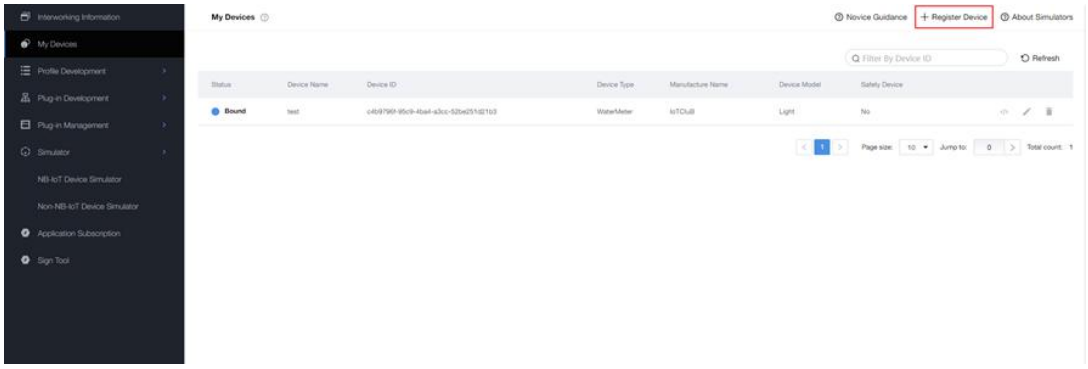
- Command 17: **AT+NMGR**. This command is used to read the issued commands.
  XXXXXX
- Command 18: **AT+NQMGS**
  PENDING=0,SENT=6,ERROR=0

  OK
- Command 19: **AT+NQMGR**
  BUFFERED=0,RECEIVED=0,DROPPED=0

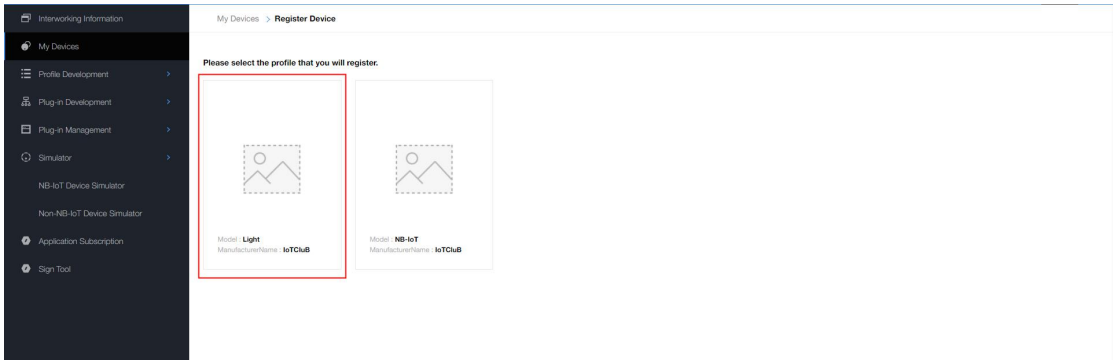
  OK

3.4.4.5 Registering an NB-IoT Device

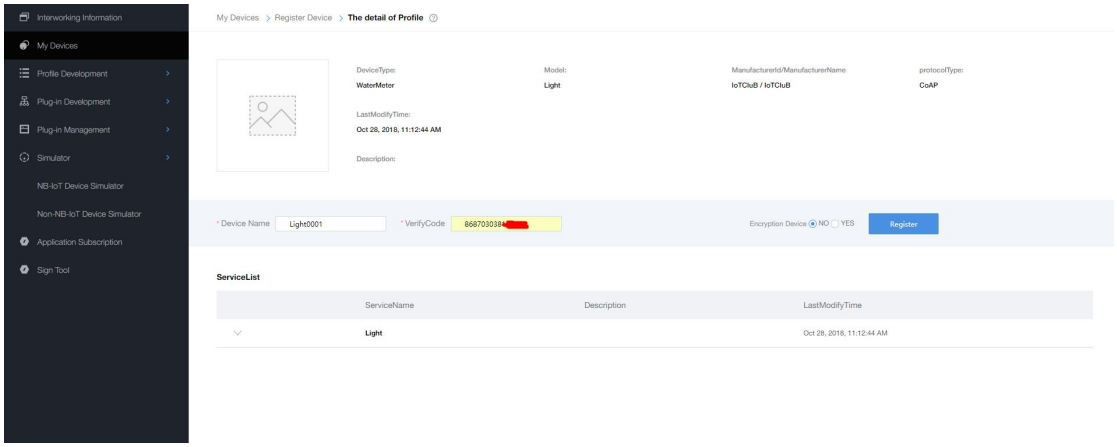
Step 1 Click **Register Device** in the upper right corner. Prepare to register a real NB-IoT device.



Step 2 Select a profile.



Step 3 Fill in the device information.



NOTE

The device ID must be the IMEI of the NB-IoT module. The IMEI can be obtained from the module or by running the **AT+CGSN=1** command. (It is recommended that the **AT+CGSN=1** command be used to read data.)

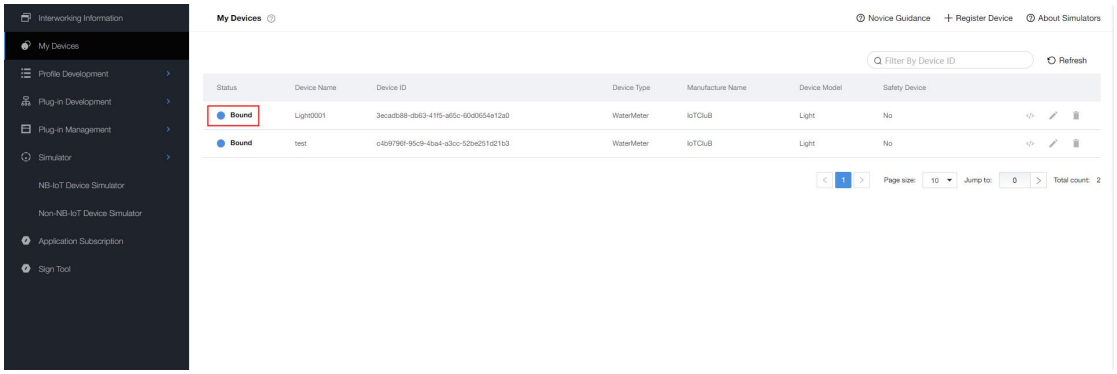
----End

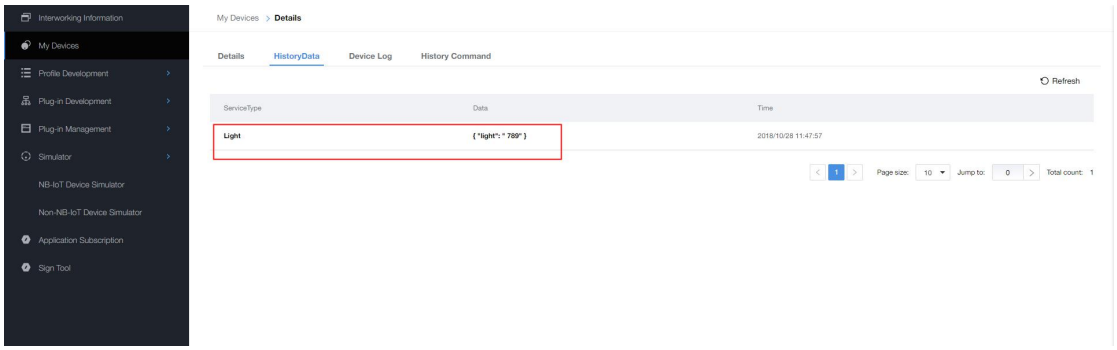
3.4.4.6 Joint Debugging of the NB-IoT and OceanConnect Platform

- Command 1: **AT+NCDP=218.4.33.72,5683**. For the IP address, see the device interconnection address of the platform.
- Command 2: **AT+NMGS=5,2020373839**. Data **789** is reported.

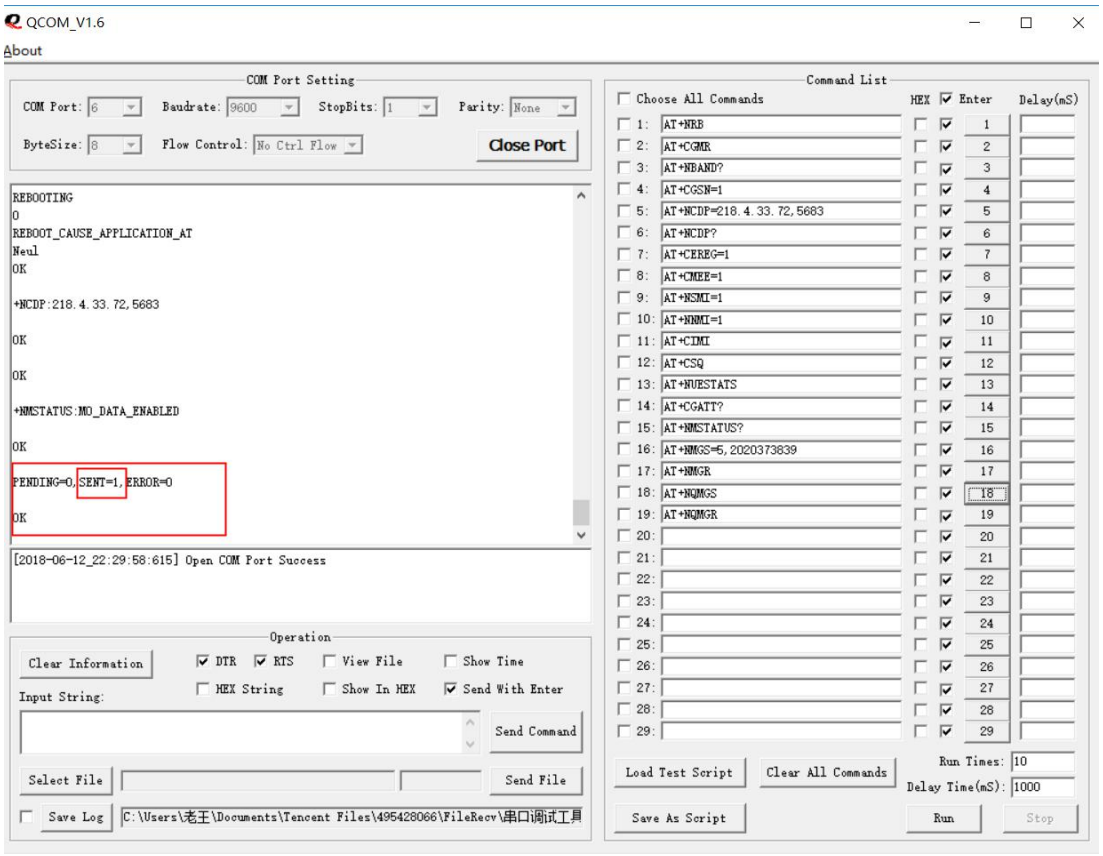
  If **ERROR** or **+CME ERROR: 513** is returned after the command is executed, send the command again 30s later after the module is restarted.

In this case, you can view the real NB-IoT device on the OceanConnect platform and bind it to the platform. Click the device to view historical data.





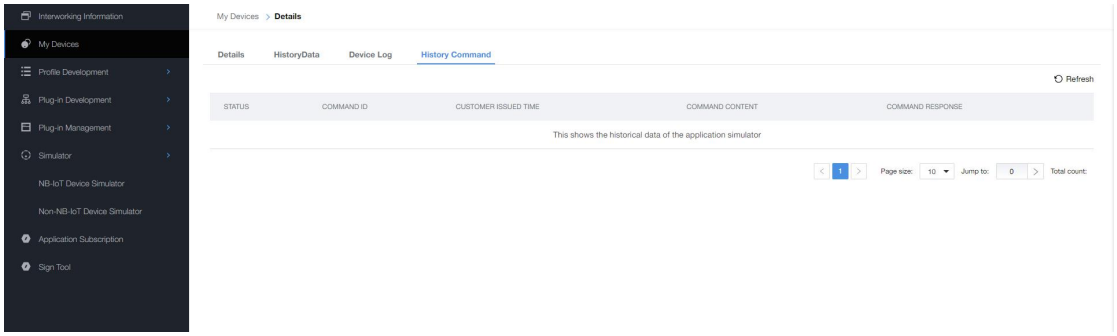
In the serial port debugging software, run the **AT+NQMGS** command and the **AT+NQMGR** command to check the statistics of sent messages and received messages.



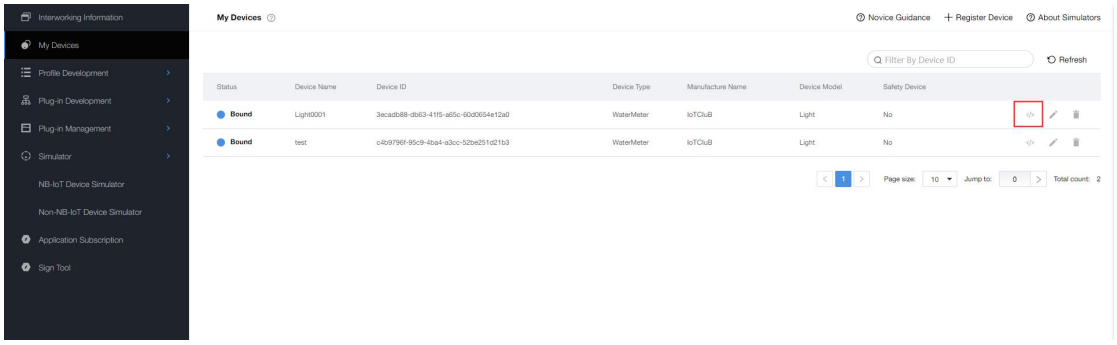
3.4.4.7 Issuing Commands from the OceanConnect to the NB-IoT Module

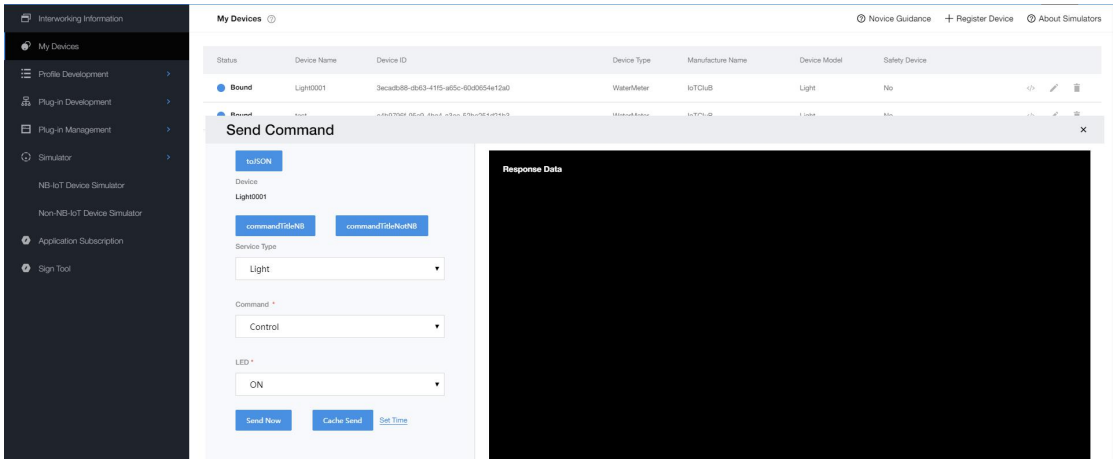
Because the NB-IoT module uses the PSM power saving mode, the OceanConnect platform does not deliver commands immediately. Instead, it waits for the NB-IoT module to send a piece of data and then delivers the commands cached on the cloud platform. Therefore, before delivering commands to the IoT cloud platform, you need to send a piece of data through the NB-IoT module to the platform. The detailed operations are as follows: Use the serial port debugging software to send the **AT+NMGS=5,2020333435** command. Then, click a command on the cloud platform to issue the command.

- On the OceanConnect platform, click a device to view its historical commands. Because no command is issued, the data is blank.

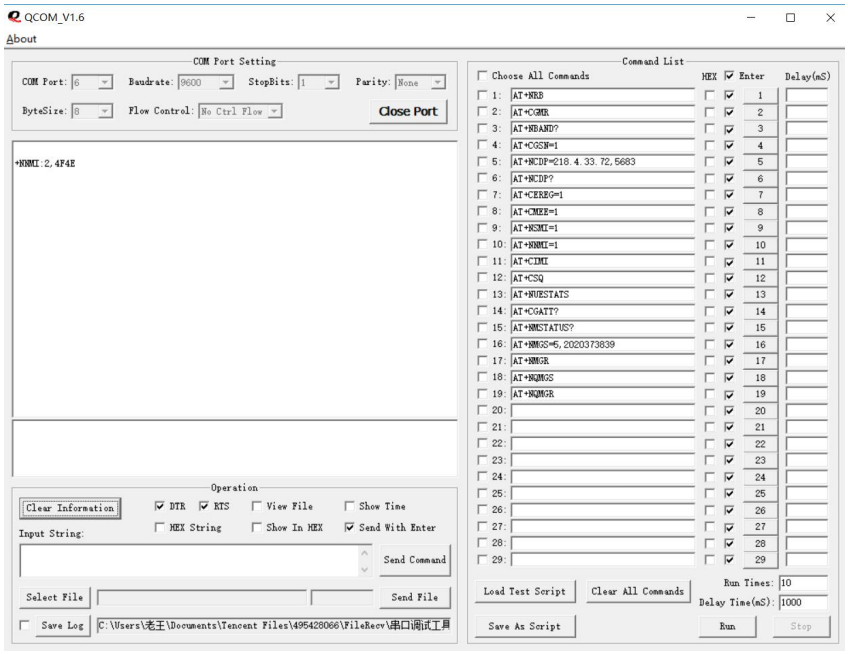


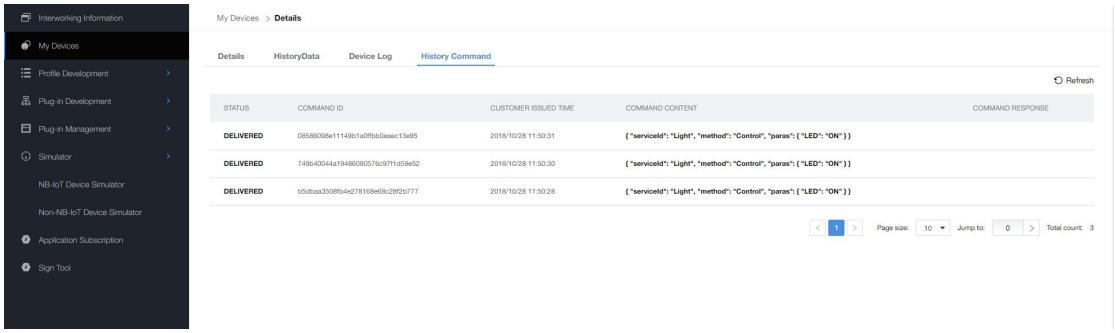
- Return to the device list. Click xx to access the command issuing page.
- Set the LED delivery control command. Before sending the command, it is recommended that you use the **AT+NMGS=5,2020333435** command to report a piece of data first. Click to send the command.





The IoT cloud platform issues a command once, and the serial port debugging assistant receives data once.



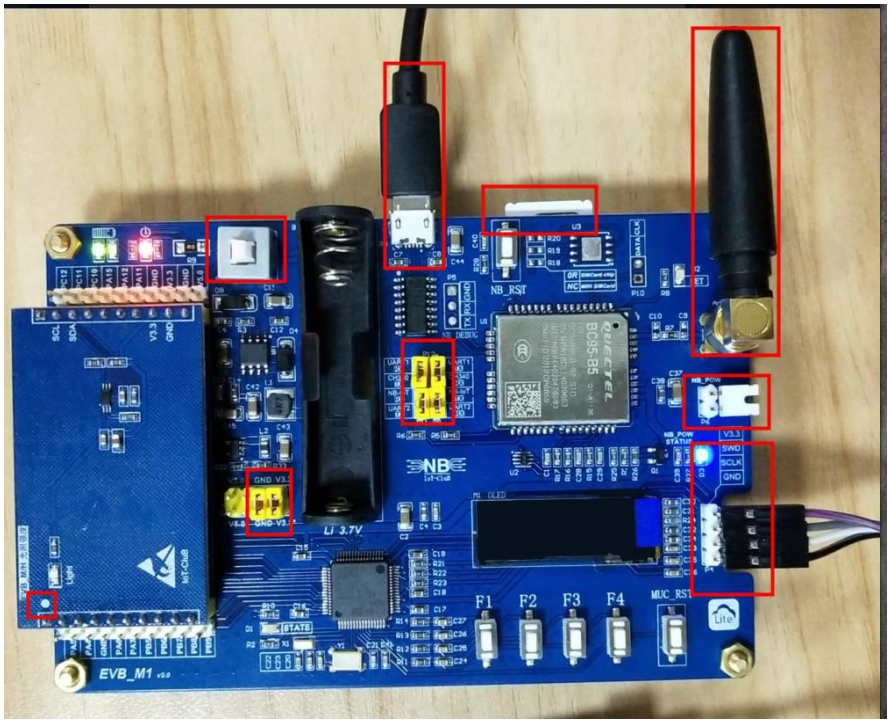


The NB-IoT module is connected to the OceanConnect platform.

3.4.5 LiteOS Debugging

3.4.5.1 Hardware Connection of the Development Board

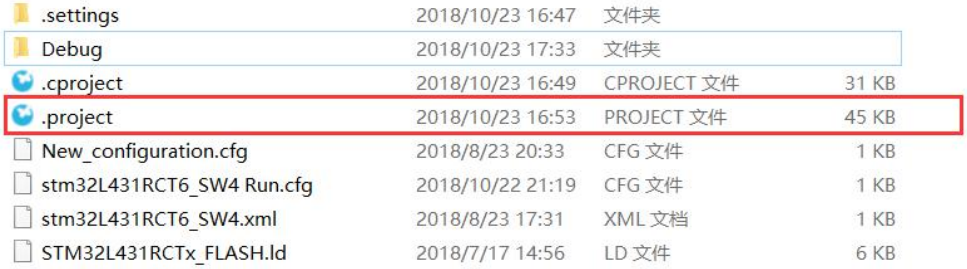
Check whether the positions in the figure are correctly connected.



Note the connection mode of the P12 jumper cap and burner.

3.4.5.2 Opening the SW4STM32 Project

Go to **targets\NB-IoT_STM32L431RxTx_IoTClub\SW4STM32\stm32L431RCT6_SW4** and open the .project file.



3.4.5.3 Modifying the Program

Step 1 Open the **mian.c** file, locate **nb_iot_entry();**, and execute the function.

main.c

```
70      OLED_ShowCHinese(90,0,5);
71      OLED_ShowCHinese(108,0,6);
72      OLED_ShowString(0,2,(uint8_t*)str,16);
73      LOS_HwiCreate(EXTI0_IRQn, 2,0,EXTI0_IRQHandler,NULL);
74      LOS_HwiCreate(EXTI1_IRQn, 3,0,EXTI1_IRQHandler,NULL);
75      LOS_HwiCreate(EXTI2_IRQn, 4,0,EXTI2_IRQHandler,NULL);
76      LOS_HwiCreate(EXTI3_IRQn, 5,0,EXTI3_IRQHandler,NULL);
77  }
78
79
80  VOID main_task(VOID)
81  {
82      user_hw_init();
83      #if defined(WITH_LINUX) || defined(WITH_LWIP)
84          hieth_hw_init();
85          net_init();
86      #elif defined(WITH_AT_FRAMEWORK) && defined(USE_NB_NEUL95)
87          nb_iot_entry();
88
89      #endif
90
91
92  }
93
```

Step 2 Change the device interconnection address of the platform based on the device interconnection address in the email sent after the platform is applied for.

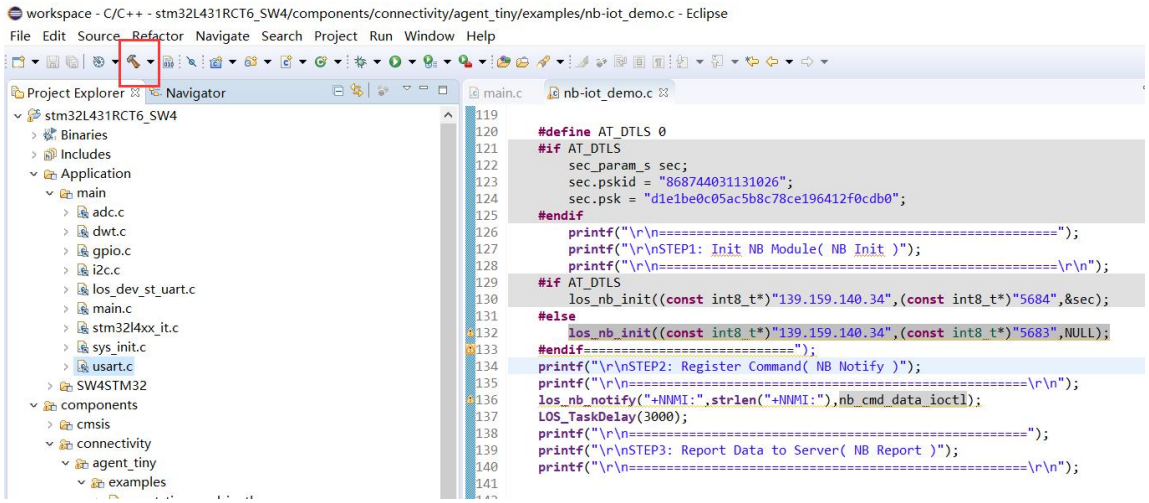
main.c nb-iot_demo.c

```
119
120     #define AT_DTLS 0
121     #if AT_DTLS
122         sec_param_s sec;
123         sec.pskid = "868744031131026";
124         sec.psk = "d1e1be0c05ac5b8c78ce196412f0cdb0";
125     #endif
126         printf("\r\n=====================================================");
127         printf("\r\nSTEP1: Init NB Module( NB Init )");
128         printf("\r\n=====================================================\r\n");
129     #if AT_DTLS
130         los_nb_init((const int8_t*)"139.159.140.34",(const int8_t*)"5684",&sec);
131     #else
132         los_nb_init((const int8_t*)"139.159.140.34",(const int8_t*)"5683",NULL);
133     #endif============================");
134     printf("\r\nSTEP2: Register Command( NB Notify )");
135     printf("\r\n=====================================================\r\n");
136     los_nb_notify("+NNMI:",strlen("+NNMI:"),nb_cmd_data_ioctl);
137     LOS_TaskDelay(3000);
138     printf("\r\n=====================================================");
139     printf("\r\nSTEP3: Report Data to Server( NB Report )");
140     printf("\r\n=====================================================\r\n");
141
142
```

----End

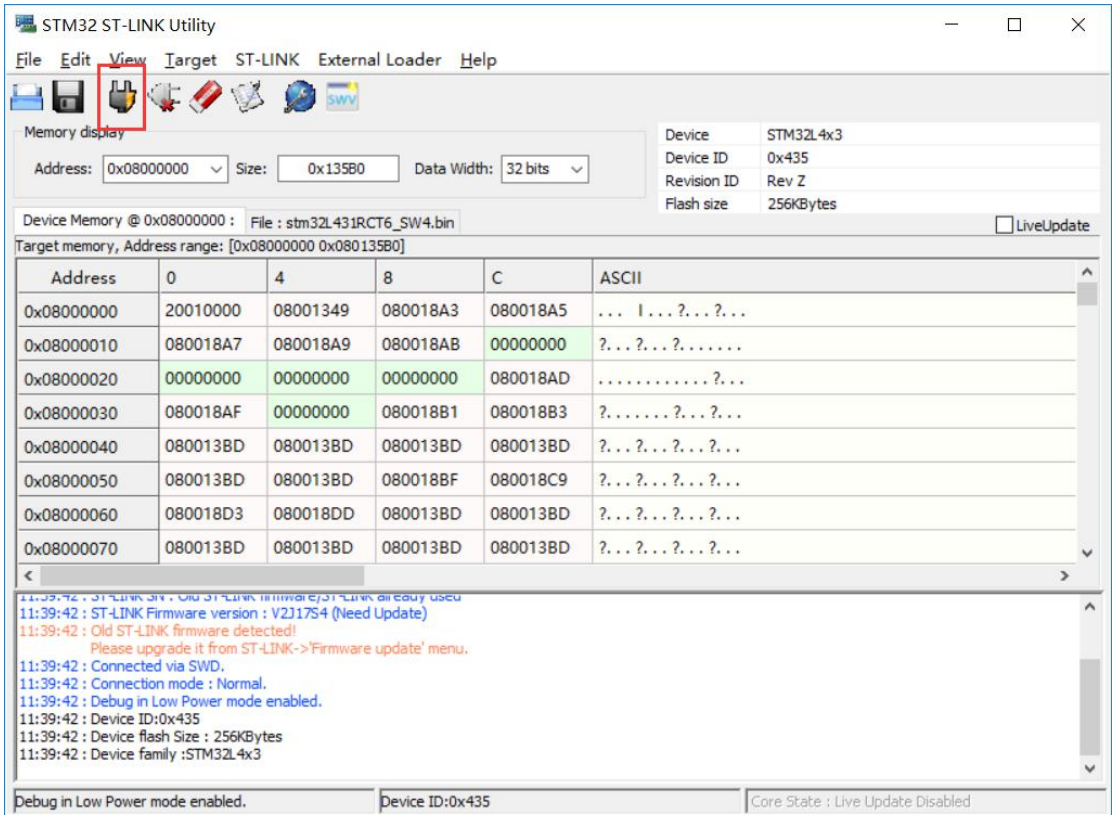
3.4.5.4 Compiling the Project

Click the compilation button to compile the project.

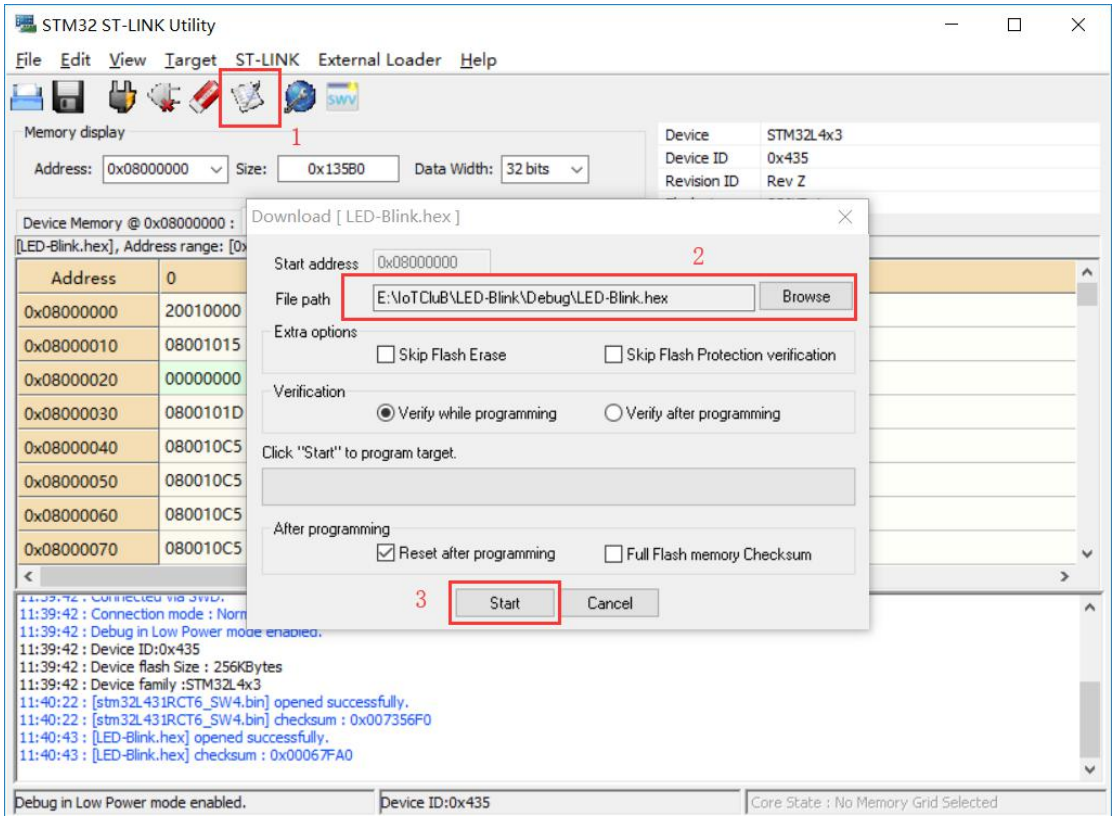


3.4.5.5 Download and run

Step 1 Open the STM32 ST-LINK Utility and connect it to the burner.

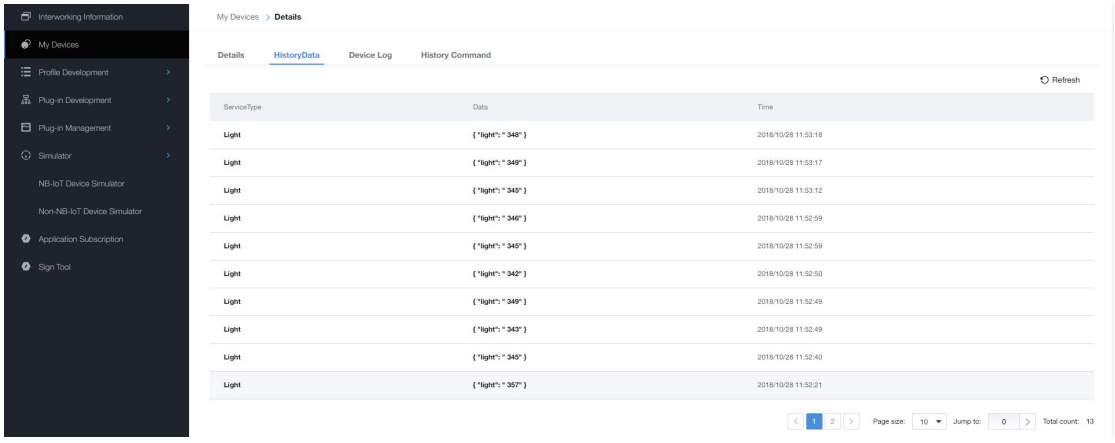


Step 2 Select the file for program burning.

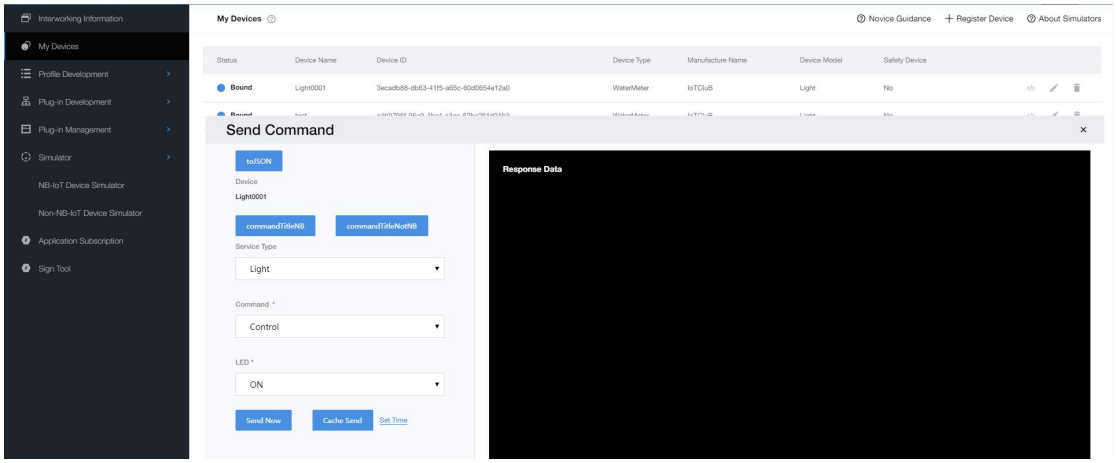


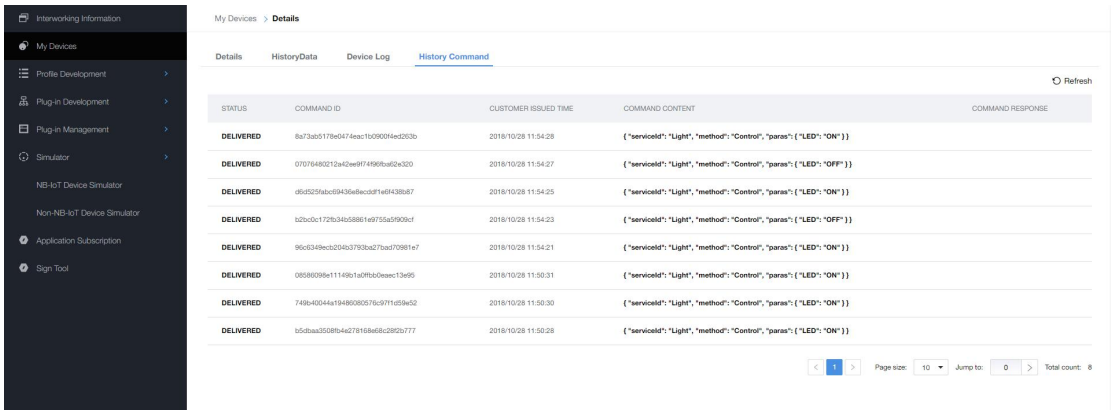
----End

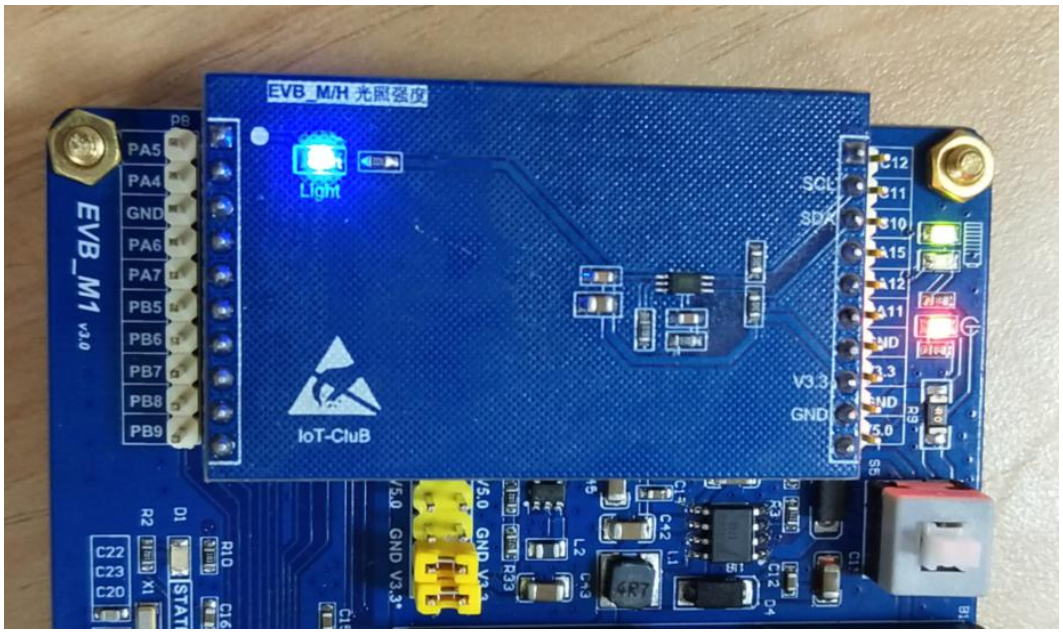
3.4.5.6 Viewing Data on the OceanConnect Platform



3.4.5.7 Issuing Commands on the OceanConnect Platform







The LiteOS OneLight debugging is complete.

3.4.6 Experiencing a Mobile App

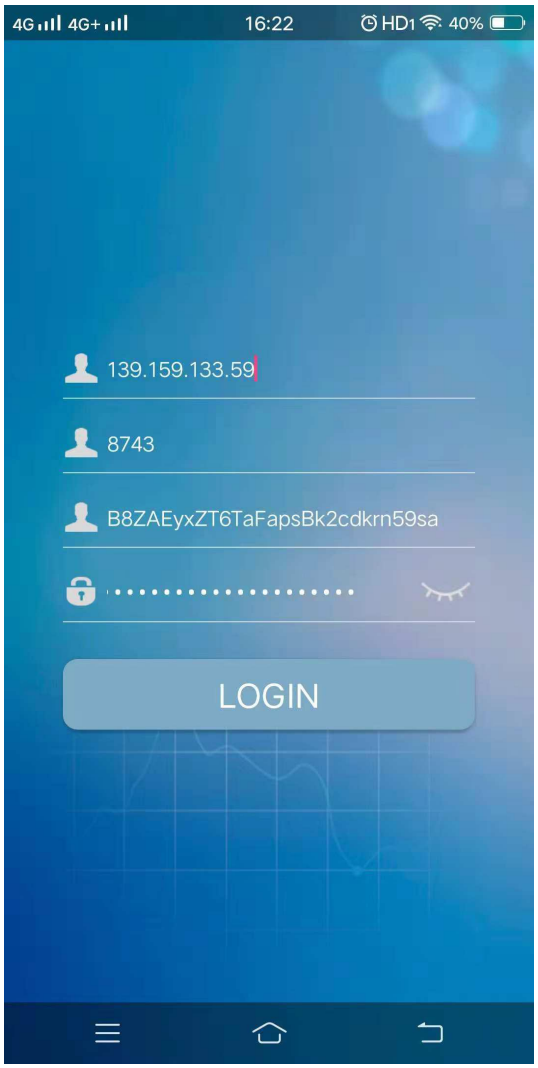
3.4.6.1 Obtain a Mobile App

Smart Street Lamp.apk

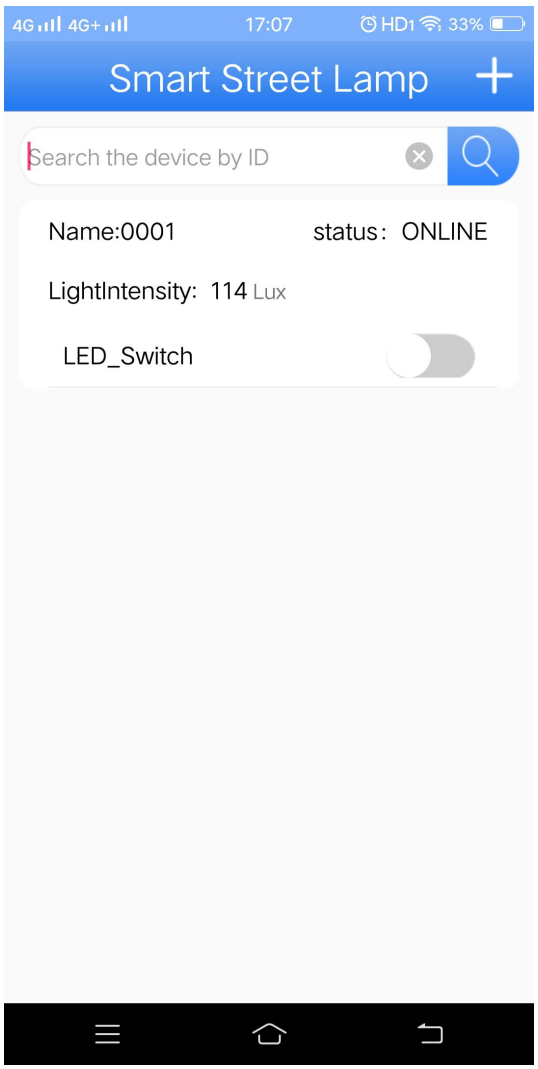
3.4.6.2 Installing the App

3.4.6.3 Logging In to the App

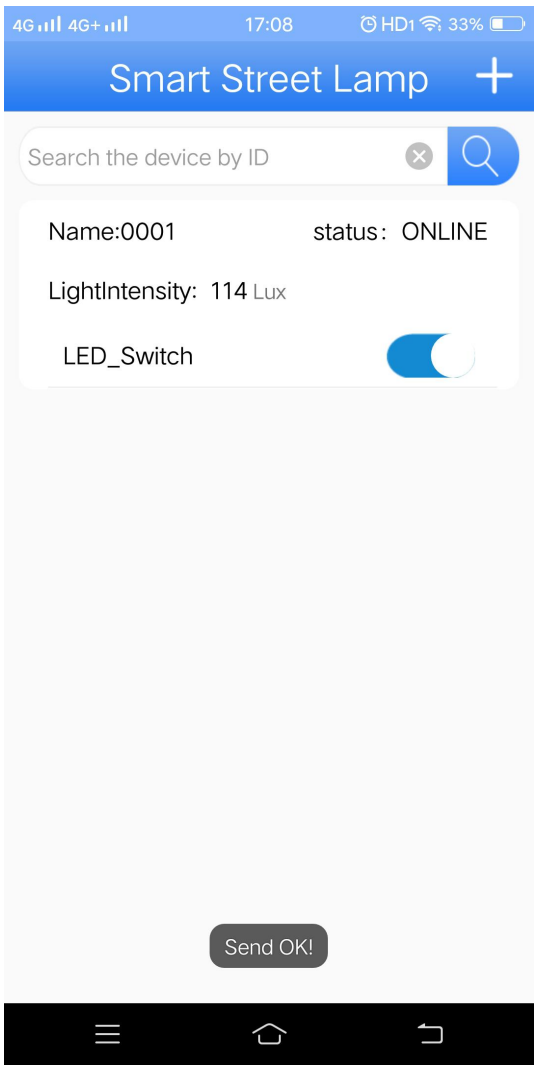
Set parameters based on the email received after an OceanConnect account is registered.



3.4.6.4 Viewing data in Real Time on the App



3.4.6.5 Issuing Commands from the App



After the issuing is successful, the indicator is on.

3.4.6.6 Additional Functions of the App

- Double-click a device to view its historical data.