# FSC-BT630 User Guide

Release 5.2.0

# Table of contents

# Chapter 1

# Introduction

## 1.1 Description

This design guide is suitable for engineers developing FSC-BT630 series Bluetooth modules

## 1.2 Module Default Settings

| | |
|---|---|
| **Name** | Feasycom |
| **Service UUID** | FFF0 |
| **Write UUID** | FFF2 |
| **Notify UUID** | FFF1 |
| **UART Baudrate** | 115200/8/N/1 |

# Chapter 2

# Hardware Description

## 2.1 Pin Diagram

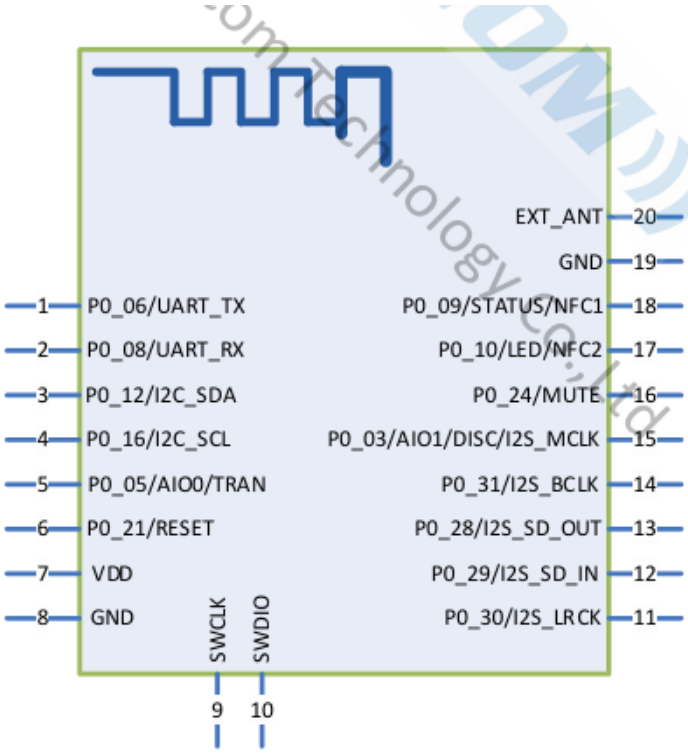



**Figure 3:** FSC-BT630 PIN Diagram(Top View)

## 2.2 Pin Description

| Pin | Pin Name | Type | Pin Descriptions |
|---|---|---|---|
| 1 | UART_TX | O | UART data output |
| 2 | UART_RX | I | UART data input |
| 6 | RESET | I | External reset input: Active LOW |
| 7 | VDD | Power | Power supply voltage 3.3V, LDO power supply preferred |
| 8 | GND | GND | GND |
| 9 | SWCLK | I/O | Serial wire debug clock input for debugand programming |
| 10 | SWDIO | I/O | Serial wire debug I/O for debug and programming |
| 17 | LED | O | Bluetooth not connected output square wave, Bluetooth connection output high level |
| 18 | STATUS | O | Bluetooth not connected output low level, Bluetooth connection output high level |
| 20 | EXT_ANT | ANT | Changing the 0 ohm resistance near the antenna, it is possible to connect a Bluetooth antenna externally |

## 2.3 Hardware Design Notes

- The simple test of the module only needs to connect VDD/GND/UART_RX/UART_TX to use
- If the MCU needs to obtain the connection status of the Bluetooth module, it needs to be connected to the STATUS pin
- After drawing the schematic diagram, please send it to Feasycom for review, so as to prevent the Bluetooth distance from not reaching the best effect
- Module supports wake-up via serial port
- VDD/GND/RESET/SWCLK/SWDIO are serial wire debug I/O, which can reserve test points

# Chapter 3

# Function Description

## 3.1 GPIO state

LED PIN is PIN 17

| State | Description |
| --- | --- |
| 2Hz square wave | disconnect state |
| high level | connected state |

Connection status PIN is PIN 18

| State | Description |
| --- | --- |
| low level | disconnect state |
| high level | connected state |

## 3.2 Mode

| | |
|---|---|
| Throughput mode | when disconnected, the data received by the serial port is analyzed according to the AT command;<br>when connected, all the data received by the serial port is sent to the remote Bluetooth as it is. |
| Command mode | when disconnected, the data received by the serial port is analyzed according to the AT command;<br>when connected, The data received by the serial port is still parsed according to the AT command.<br>when it is necessary to send data to the remote end, send AT+LESEND command. |

## 3.3 GATT Throughput Service

| type | UUID | Permissions | Description |
|---|---|---|---|
| Service | 0xFFF0 | | GATT Throughput Service |
| Write | 0xFFF2 | Write， Write Without Response | APP send data to module |
| Notify | 0xFFF1 | Notify | Module send data to remote |

## 3.4 Low Power

Support 1 low power consumption mode, serial wake-up mode.

Table 1: Serial wake-up mode.

| Mode | AT command | Sleep method | Description |
|---|---|---|---|
| Serial wake-up | AT+LPM=1 | If the serial port exceeds 5s and there is no data communication, it will automatically enter sleep. After sleep, the serial port will exit sleep mode after receiving the first frame of data. | The first frame of data that wakes up will be lost. Simple logic, saves IO. |

# Chapter 4

# AT command description

## 4.1 Command description

Throughout this specification:

- Content between { } is optional
- Content behind << represents a **COMMAND** from Host
- Content behind >> represents a **RESPONSE/EVENT** to Host

## 4.2 Command

The command is the control command sent by the host to the module actively. After receiving the command, the module needs to reply **<CR><LF>OK<CR><LF>** as a response.

## 4.3 Command Format

**AT+Command{=Param1{,Param2{,Param3…}}}<CR><LF>**

- All commands start with **AT**, end with **<CR><LF>**
- **<CR>** means “carriage return”, corresponds to hex value 0x0D
- **<LF>** means “line feed”, corresponds to hex value 0x0A

- If Command has Parameter, Parameter follows behind ‘=’
- If Command has multiple Parameters, Parameter must be separated by ‘,’
- If Command has Response, Response starts with **<CR><LF>**, ends with **<CR><LF>**
- The module should always return the result of instruction execution (success returns **OK**，failure returns **ERROR**)

Example:

Read module’s name

<< AT+VER

>> +VER=5.2.0,FSC-BT630

>> OK

Write invalid baudrate

<< AT+BAUD=0

>> ERROR

## 4.4 Event

Event is the data that the module actively sends to the host. Generally used to indicate a change of state or data received.

## 4.5 Event Format

**<CR><LF>+Indication{=Param1{,Param2{,Param3…}}}<CR><LF>**

- All Events start with <CR><LF>, end with <CR><LF>
- If Event has Parameter, Parameter follow behind ‘=’
- If Event has multiple Parameters, Parameter must be separated by ‘,’

Example:

Receive data from APP in command mode

>> +DATA=10,1234567890

# Chapter 5

# Commands Table

## 5.1 AT - Serial communication test

| Command | AT |
|---|---|
| **Response** | **OK** |
| **Description** | When powering on or changing the baud rate, test the UART communication between the host and the module |

Example:

<< AT

>> OK

## 5.2 AT+VER - Get Firmware Version

| Command | AT+VER |
|---|---|
| **Response** | **+VER=Param** |
| `Param` | Firmware version |
| **Description** | Get Firmware Version |

Example:

<< AT+VER

>> +VER=5.2.0,FSC-BT630

>> OK

## 5.3 AT+ADDR - Get MAC Address

| **Command** | **AT+ADDR** |
|---|---|
| **Response** | **+ADDR=Param** |
| `Param` | Module’s MAC address (12 Bytes ASCII) |
| **Description** | Get MAC Address |

Example:

<< AT+ADDR

>> +ADDR=DC0D30010203

>> OK

## 5.4 AT+NAME - Get/Set Local Name

| **Command** | **AT+NAME{=Param1{,Param2}}** |
|---|---|
| Param1 | local name(1~29 Bytes ASCII) |
| Param2 | MAC address suffix(0/1,default:0)<br>0: Disable suffix<br>1: Enable suffix “XXXX”(lower 4 bytes of MAC address) after local name |
| **Response** | **+NAME=Param** |
| Param | local name |
| **Description** | Write local name if parameter exist, otherwise read current local name |

Example:

Read current local name

<< AT+NAME

>> +NAME=Feasycom

>> OK

Change module’s local name to “ABC”

<< AT+NAME=ABC

>> OK

Change module’s local name to “ABC”and enable suffix

<< AT+NAME=ABC,1

>> OK

## 5.5 AT+BAUD - Get/Set Uart Baudrate

| **Command** | **AT+BAUD{=Param}** |
|---|---|
| `Param` | baudrate(2400/4800/9600/14400/19200/28800/38400/ 57600/115200/230400/460800/500000/921600/1000000, default:115200) |
| **Response** | **+BAUD=Param** |
| `Param` | baudrate |
| **Description** | Module will change baudrate to target value immediately |

Example:

Query baudrate

<< AT+BAUD

>> +BAUD=115200

>> OK

Change baudrate

<< AT+BAUD=9600

>> OK

## 5.6 AT+TXPOWER - Get/Set TX Power

| **Command** | **AT+TXPOWER{=Param}** |
|---|---|
| Param | tx power(0~9, default:0)<br>0: 0dbm<br>1: -40dbm<br>2: -20dbm<br>3: -16dbm<br>4: -12dbm<br>5: -8dbm<br>6: -4dbm<br>7: 0dbm<br>8: 3dbm<br>9: 4dbm |
| **Response** | **+TXPOWER=Param** |
| Param | tx power |
| **Description** | Get/Set TX Power |

Example:

Get tx powe

<< AT+TXPOWER

>> +TXPOWER=7

>> OK

Set tx power to 4dbm

<< AT+TXPOWER=9

>> OK

## 5.7 AT+LPM - Get/Set low power config

| **Command** | **AT+LPM{=Param}** |
|---|---|
| `Param` | low power mode (0/1, default: 0)<br>0: disable<br>1: enable low power and wake up by UART |
| **Response** | **+LPM=Param** |
| `Param` | low power mode |
| **Description** | Get/Set low power config |

Example:

Get low power config

<< AT+LPM

>> +LPM=0

>> OK

Set serial port wakeup mode

<< AT+LPM=1

>> OK

## 5.8 AT+ADVIN - Get/Set advertising interval

| **Command** | **AT+ADVIN{=Param}** |
|---|---|
| `Param` | advertising interval (200~10000 ms0 ms, 默认: 200ms) |
| **Response** | **+ADVIN=Param** |
| `Param` | advertising interval |
| **Description** | Get/Set advertising interval |

Example:

Get advertising interval

<< AT+ADVIN

>> +ADVIN=152

>> OK

Set advertising interval to 100ms

<< AT+ADVIN=100

>> OK

## 5.9 AT+LEDISC - Disconnect specified connection

| **Command** | **AT+DISC** |
|---|---|
| `Param` | Connection serial number (disconnect all connections by default) |
| **Response** | **OK** |
| **Description** | disconnect Bluetooth connection |

Example:

disconnect all device

<< AT+LEDISC

>> OK

## 5.10 AT+ADVDATA - Get/Set advertising manufacturer specific data

| **Command** | **AT+ADVDATA{=Param}** |
|---|---|
| `Param` | manufacturer specific data, tag 0xff |
| **Response** | **+ADVDATA=Param** |
| `Param` | manufacturer specific data, tag 0xff |
| **Description** | Get/Set advertising manufacturer specific data |

Example:

Get advertising manufacturer specific data, \x is hexadecimal data

<< AT+ADVDATA

>> +ADVDATA=\x03\x01\x02

>> OK

Set advertising manufacturer specific data, \x is hexadecimal data

<< AT+ADVDATA=\x03\x01\x02

>> OK

## 5.11 AT+IBEACON - Get/Set advertising manufacturer specific data swhitch

| **Command** | **AT+IBEACON{=Param}** |
|---|---|
| Param | （0~1， Default:1）<br>0: turn off manufacturer specific data, tag 0xff<br>1: turn on manufacturer specific data, tag 0xff |
| **Response** | **+IBEACON=Param** |
| **Description** | When turned on, BLE will switch between the two broadcast data |

Example:

turn on manufacturer specific data, tag 0xff

<< AT+IBEACON=1

>> OK

turn off manufacturer specific data, tag 0xff

<< AT+IBEACON=0

>> OK

## 5.12 AT+LESEND - Send data to remote device

| Command | AT+LESEND{=Param1,Param2} |
|---|---|
| Param1 | communication channel |
| Param2 | data length |
| Param3 | data |
| **Response** | **OK** |
| **Description** | Even the index information is queried using the AT+CHINFO command |

Example:

Send data to the connected device

<< AT+LESEND=1,4,2022

>> OK

## 5.13 AT+REBOOT - Soft Reboot

| Command | AT+REBOOT |
|---|---|
| **Response** | **OK** |
| **Description** | Module release all Bluetooth connections then reboot |

Example:

<< AT+REBOOT

>> OK

## 5.14 AT+RESTORE - Restore Factory Settings

| **Command** | **AT+RESTORE** |
|---|---|
| **Response** | **OK** |
| **Description** | Module restore all factory settings then reboot |

Example:

<< AT+RESTORE

>> OK

## 5.15 AT+PIOCFG - Get/Set Pin Function Switch

| **Command** | **AT+PIOCFG{=param1,param2}** |
|---|---|
| `param1` | Working mode (0~ 1, default 0). PIN5 low-level throughput mode, high-level command mode.<br>0: Disable<br>1: Enable |
| `param2` | Disconnect (0~ 1, default 1). PIN14 falling edge disconnects.<br>0: Disable<br>1: Enable |
| **Response** | **+PIOCFG=Param1,Param2** |
| **Description** | Modify Pin function |

Example:

Turn on pin control mode

<< AT+PIOCFG=1

>> OK

Turn off the function of two pins.

<< AT+PIOCFG=0,0

>> OK

## 5.16 AT+REGUUID - Register 128bit UUID with the protocol stack

| **Command** | **AT+REGUUID=Param** |
|---|---|
| `Param2` | UUID（32-Byte hex string） |
| **Response** | **OK** |
| **Description** | 128Bit UUID requires registration to be recognized and may be used when connecting devices |

Example:

Register128bit UUID：00001100d10211e19b2300025b00a5a5

<< AT+REGUUID=00001100d10211e19b2300025b00a5a5

>> OK

## 5.17 AT+LECCONN - Initiate a connection to the specified address

| **Command** | **AT+LECCONN{=Param1{,Param2{,Param3{,Param4}}}}** |
|---|---|
| Param1 | 12-byte device address + 1-byte address type |
| Param2 | Communication Service UUID |
| Param3 | Communication UUID with write/write without rsp permission |
| Param4 | Communication UUID with notification permission |
| **Response** | **OK** |
| **Description** | Initiate a connection to the specified device, the parameter consists of 12 bytes (device address) and 1 byte (address type), generally the address type is “0” or “1”. address How to get the address type: Send AT+SCAN=1 command, get parameter2. example: +SCAN=0,0,DC0D30001ED4,-65,10,FSC-BT946 Connection command: AT+LECCONN=DC0D30001ED40 |

Example:

Connect to the specified device, 0 is the address type

<< AT+LECCONN=DC0D3000039E0

>> OK

Initiate a connection to the specified address, using FFF0, FFF2, FFF1 for communication

<< AT+LECCONN=DC0D3000039E0,FFF0,FFF2,FFF1

>> OK

## 5.18 AT+SCAN - SCAN for nearby devices

| **Command** | **AT+SCAN{=Param1{, Param2{, Param3}}}** |
|---|---|
| Param1 | scanning method（0~1）<br>0: stop scanning<br>1: Scan nearby devices, filter duplicate addresses, up to 8 devices can be found<br>2: Scan the BLE advertising packet and output the original advertising data packet |
| **Response** | **+SCAN=Param1,Param2,Param3,Param4,Param5,Param6**<br>(AT+SCAN=1)<br><br>**+SCAN=Param21,Param22,Param23,Param24,Param25,Param62**<br>(AT+SCAN=2) |
| Param6 | Device index |
| Param1 | MAC Address type (1 byte) |
| Param2 | MAC address (12 bytes) |
| Param3 | RSSI |
| Param4 | Device name length |
| Param5 | Device name |
| Param21 | MAC Address type (1 byte) |
| Param22 | MAC address (12 bytes) |
| Param23 | RSSI |
| Param24 | Ad Type |
| Param25 | Ad data length |
| Param26 | Ad data package（Hex） |
| **Description** | AT+SCAN=1 Commonly used for searching before connecting<br>AT+SCAN=2 used as a Bluetooth gateway to search for nearby Beacon devices |

Example:

SCAN device

<< AT+SCAN=1

>> +SCAN={

>> +SCAN=1,70CFC9A98840,-43,24,LE-Bose QuietControl 30

>> +SCAN=1,DC0D30001ED4,-65,10,FSC-BT946

>> +SCAN=}

Sniff BLE ads and return data

<< AT+SCAN=2

>> +SCAN{

>> +SCAN=0,DC0D3000129D,-48,0,40,020106030328180C1628180054D621E1DD167776

>> +SCAN}

## 5.19 AT+CHINFO - Read connection peer information

| **Command** | **AT+CHINFO=Param** |
|---|---|
| Param | 0：stop report；1：report once；2：report regularly one second |
| **Response** | **+CHINFO=Param1,Param2,Param3,Param4,Param5,Param6,Param7,Param8** |
| Param1 | connected index |
| Param2 | connection status (1：Standby，2：Connecting，3：Connected) |
| Param3 | Connect role (0：Server，1：Client) |
| Param4 | Peer address |
| Param5 | RSSI |
| Param6 | connection interval |
| Param7 | Server delay |
| Param8 | connection timedout |
| **Description** | Query connection information |

Example:

Query connection information

<< AT+CHINFO

>> +CHINFO{

>> +CHINFO=0,3,1,50AE7CE23D14,-54,48,1,400

>> +CHINFO=1,1,0,000000000000,0,0,0,0

>> +CHINFO=2,1,0,000000000000,0,0,0,0

>> +CHINFO=3,1,0,000000000000,0,0,0,0

>> +CHINFO=4,1,0,000000000000,0,0,0,0

>> +CHINFO=5,1,0,000000000000,0,0,0,0

>> +CHINFO}

>> OK

## 5.20 AT+UUIDCFG - Configure the UUID for throughput mode

| **Command** | **AT+UUIDCFG=Param1,Param2, Param3** |
|---|---|
| `Param1` | SERVER UUID，Supports 16-bit and 128-bit (hex string) |
| `Param2` | write UUID，Supports 16-bit and 128-bit (hex string) |
| `Param3` | notify UUID，Supports 16-bit and 128-bit (hex string) |
| **Response** | **OK** |
| **Description** | Configure the UUID for throughput mode |

Example:

<< AT+UUIDCFG=FFF0,FFF2,FFF1

>> OK

<<

AT+UUIDCFG=6e400001b5a3f393e0a9e50e24dcca9e,6e400003b5a3f393e0a9e50e24dcca9e,6e400002b5a3

>> OK

# Chapter 6

# Event Table

## 6.1 +SCAN - AT+SCAN=1 Scan Result

| **Indication** | **+SCAN=Param1,Param2,Param3,Param4,Param5,Param6** |
|---|---|
| Param1 | Devices index |
| Param2 | MAC address type (1 Byte) |
| Param3 | MAC address (12 Bytes ASCII) |
| Param4 | RSSI |
| Param5 | remote device name len |
| Param6 | remote device name |
| **Description** | After the module receives AT+SCAN=1, it will continue to scan, and report through +SCAN after finding the device |

Example:

SCAN device

<< AT+SCAN=1

>> +SCAN={

>> +SCAN=1,70CFC9A98840,-43,24,LE-Bose QuietControl 30

>> +SCAN=1,DC0D30001ED4,-65,10,FSC-BT946

>> +SCAN=}

## 6.2 +SCAN - AT+SCAN=2 Scan Result

| **Indication** | **+SCAN=Param1,Param2,Param3,Param4,Param5,Param6** |
|---|---|
| `Param1` | MAC address type（1-Byte） |
| `Param2` | MAC address（12-Byte） |
| `Param3` | RSSI |
| `Param4` | Ad type |
| `Param5` | Ad data length |
| `Param6` | Ad data package（hex string） |
| **Description** | After the module receives AT+SCAN=2, it will continue to scan, and report through +ADVDATA after finding the device |

Example:

Sniff BLE ads and return the original data source (garbled part is the original data source, using hex format)

<< AT+SCAN=2

>> +SCAN{

>> +SCAN=0,DC0D3000129D,-48,0,40,020106030328180C1628180054D621E1DD167776

>> +SCAN}

## 6.3 +GATTSTAT - GATT State

| Indication | +GATTSTAT=Param1,Param2 |
|---|---|
| Param1 | Connection channel |
| Param2 | GATT State (1~3)<br>1: Standby<br>2: Connecting<br>3: Connected |
| Description | In the command mode, if the connection status of the module changes, it will be actively reported through +GATTSTAT |

Example:

Connected

>> +GATTSTAT=0,3

## 6.4 +DATA - GATT Received Incoming Data

| Indication | +DATA=Param1,Param2,Param3 |
|---|---|
| Param1 | Connect index |
| Param2 | Payload length |
| Param3 | Payload |

Example:

received data 1234567890

>> +DATA=0,10,1234567890

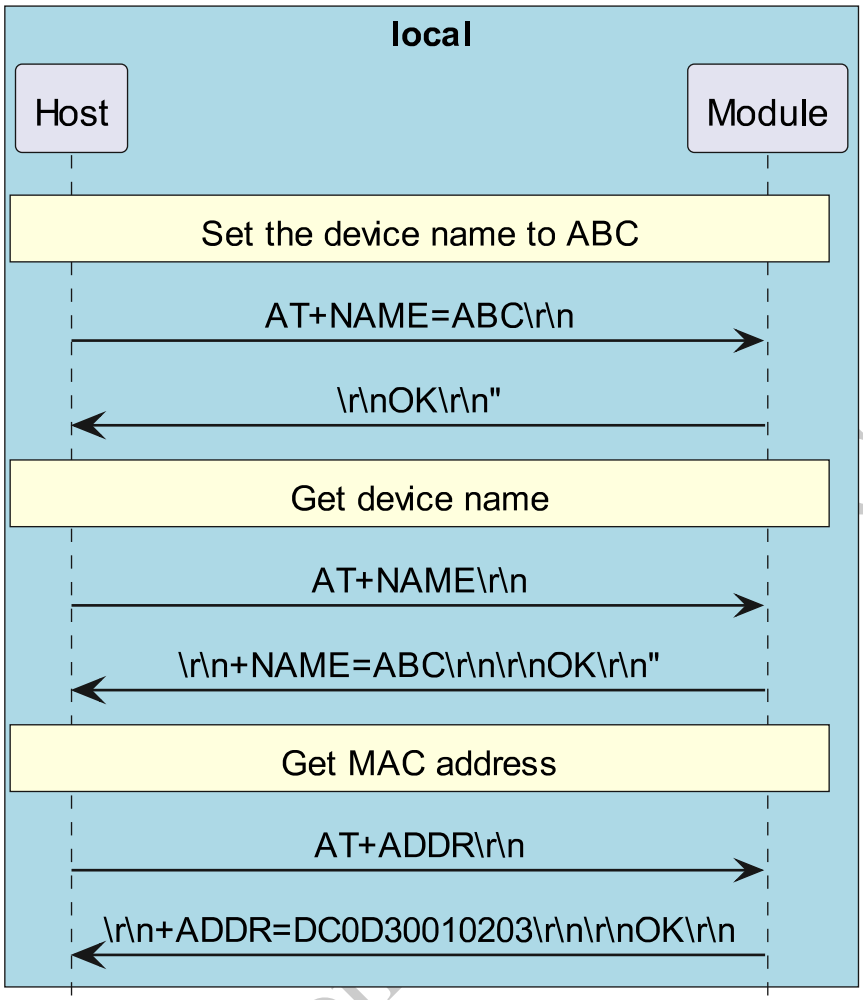
# Chapter 7

# Application scenarios

## 7.1 Get/Set the default parameters of the module

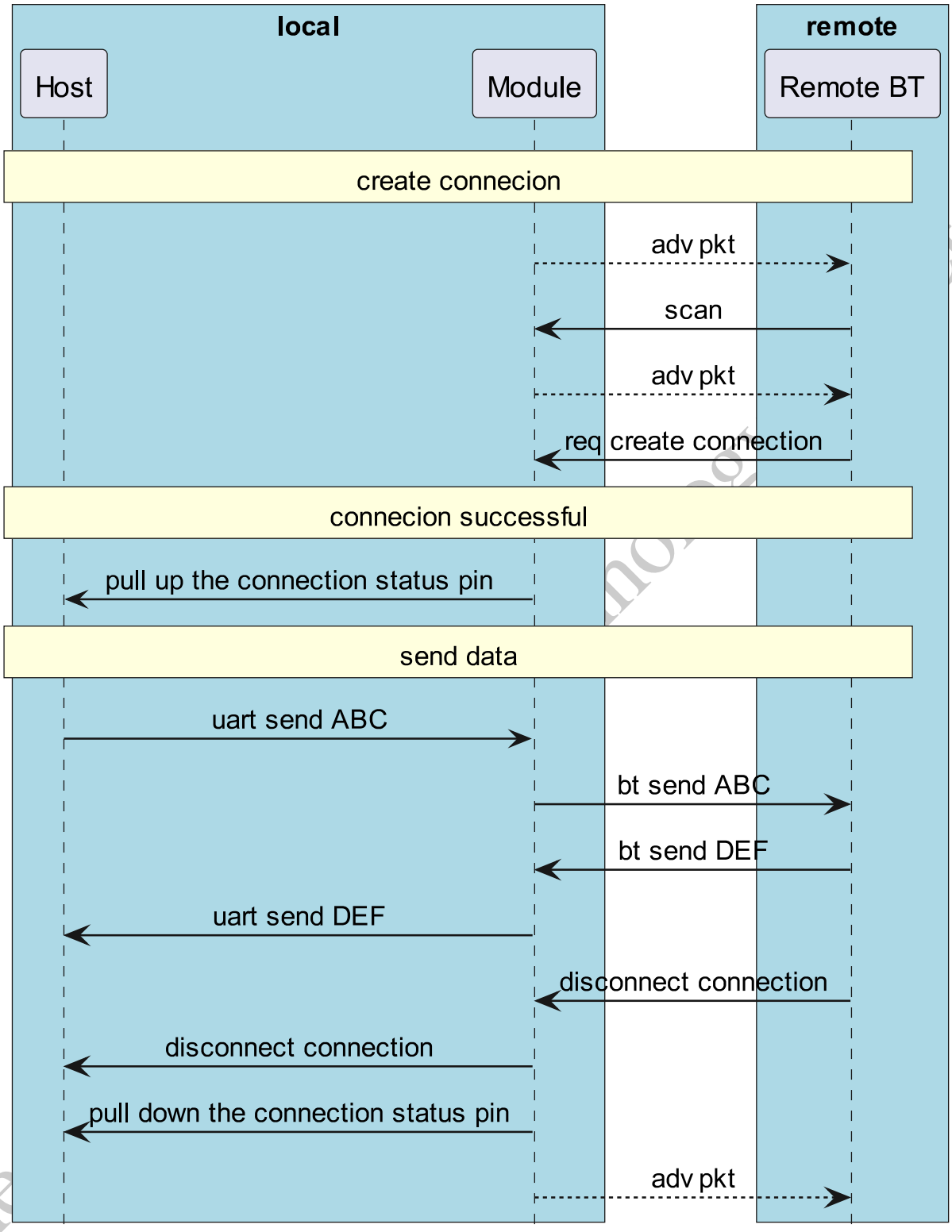
When the module is disconnect state, it will parse the uart data according to the AT command. The host can get and set the default parameters of the module, as shown in the figure below:

1. Set the device name to ABC
2. Get device name
3. Get MAC address



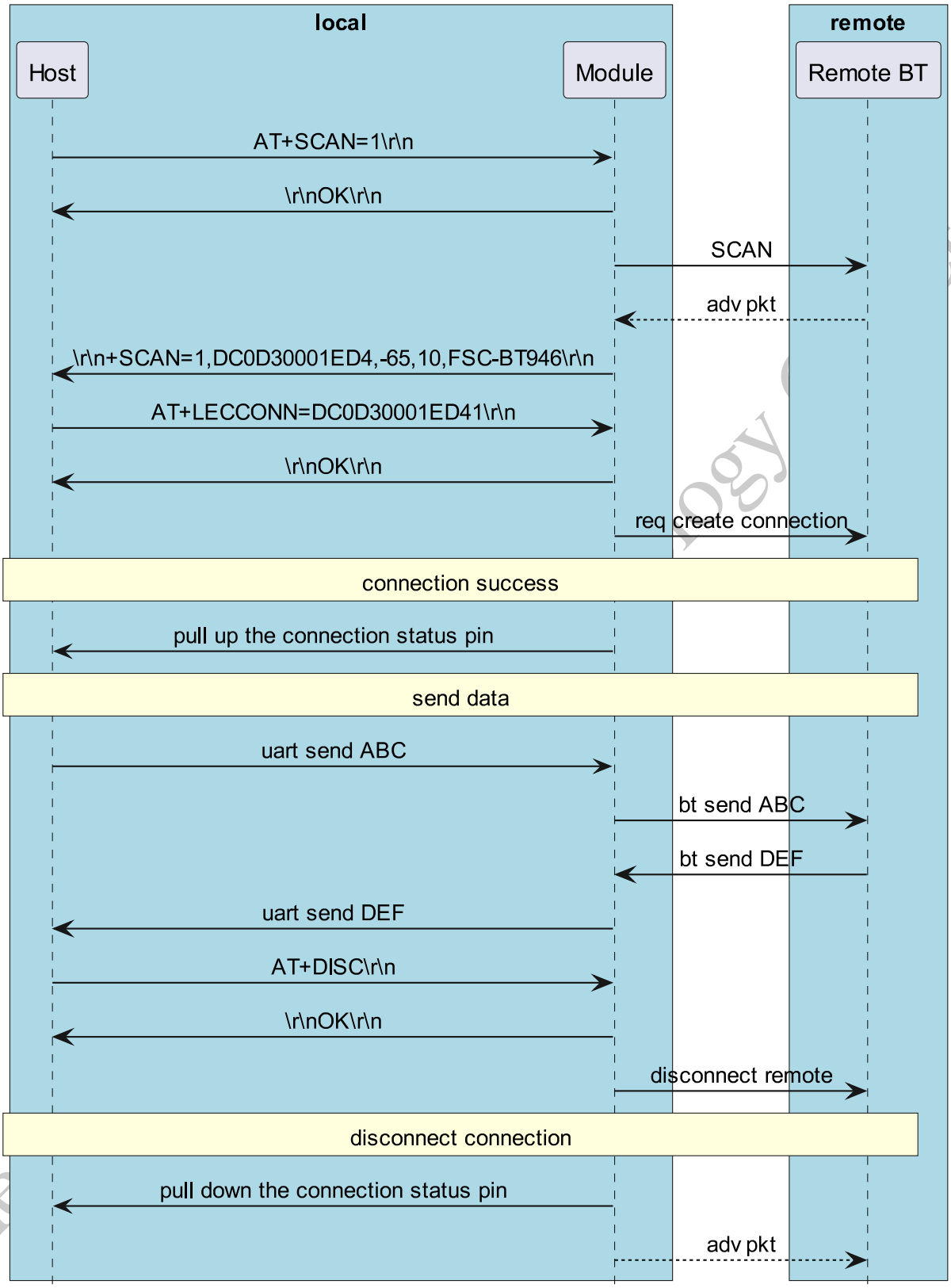

## 7.2 Process of sending data

When the module is powered on, it will continue to send advertising data, and the remote device (mobile phone) can scan the advertising packet. And initiate a connection request to the module. After the connection is successful, the module will pull up the connection status pin to notify the host that the Bluetooth connection is successful. The host can send data to the remote device through the Bluetooth module, and the remote device can also send data to the host.


local
remote
Host
Module
Remote BT
create connecion
adv pkt
scan
adv pkt
req create connection
connecion successful
pull up the connection status pin
send data
uart send ABC
bt send ABC
bt send DEF
uart send DEF
disconnect connection
disconnect connection
pull down the connection status pin
adv pkt

## 7.3 The module acts as the central to connect to the remote device

The module can be used as a master device to connect to the slave device, and the host can send commands to control the module to scan and connect and disconnect. The figure below shows the process of connecting other devices:


local
remote
Host
Module
Remote BT
AT+SCAN=1\r\n
\r\nOK\r\n
SCAN
adv pkt
\r\n+SCAN=1,DC0D30001ED4,-65,10,FSC-BT946\r\n
AT+LECCONN=DC0D30001ED41\r\n
\r\nOK\r\n
req create connection
connection success
pull up the connection status pin
send data
uart send ABC
bt send ABC
bt send DEF
uart send DEF
AT+DISC\r\n
\r\nOK\r\n
disconnect remote
disconnect connection
pull down the connection status pin
adv pkt

# Chapter 8

# FAQ

## 8.1 How can IOS phones obtain Bluetooth MAC addresses?

For security reasons, the IOS system converts the Bluetooth MAC address into UUID at the bottom layer

and sends it to the upper layer application. So the app cannot obtain the MAC address of the device.

FSC-BT630 will place the MAC address in the broadcast by default, and the APP can obtain the MAC address from the broadcast packet using the following method.

```
- (void)centralManager:(CBCentralManager *)central␣
↪didDiscoverPeripheral:(CBPeripheral *)peripheral␣
↪advertisementData:(NSDictionary *)advertisementData␣
↪RSSI:(NSNumber *)RSSI
{
    if(![self describeDictonary:advertisementData])
    {
        NSLog(@"is not fsc module");
        return;
    }
}

- (Boolean)describeDictonary: (NSDictionary *) dict
{
```

(continues on next page)

(continued from previous page)

```
    NSArray *keys;
    id key;
    keys = [dict allKeys];
    for(int i = 0; i < [keys count]; i++)
    {
        key = [keys objectAtIndex:i];
        if([key isEqualToString:@
↪"kCBAdvDataManufacturerData"])
        {
            NSData *tempValue = [dict objectForKey:key];
            const Byte *tempByte = [tempValue bytes];
            if([tempValue length] == 6)
            {
                // tempByte MAC address
                return true
            }
        }else if([key isEqualToString:@"kCBAdvDataLocalName
↪"])
        {
            //there is name
            //NSString *szName = [dict objectForKey: key];
        }
    }
    return false;
}
```

# Chapter 9

# Appendix

## 9.1 Download PDF Document

Download PDF Document