



## **GR55xx AT Command Example Application**

**Version: 1.1**

**Release Date: 2021-04-20**

**Copyright © 2021 Shenzhen Goodix Technology Co., Ltd. All rights reserved.**

Any excerption, backup, modification, translation, transmission or commercial use of this document or any portion of this document, in any form or by any means, without the prior written consent of Shenzhen Goodix Technology Co., Ltd is prohibited.

## **Trademarks and Permissions**

**GOODiX** and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Disclaimer**

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

## **Shenzhen Goodix Technology Co., Ltd.**

Headquarters: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828

FAX: +86-755-33338099

Website: [www.goodix.com](http://www.goodix.com)

## Preface

### Purpose

This document introduces how to use and verify the ble\_app\_uart\_at example in the GR55xx SDK, to help users quickly get started with secondary development.

### Audience

This document is intended for:

- GR55xx user
- GR55xx developer
- GR55xx tester
- Hobbyist developer

### Release Notes

This document is the second release of *GR55xx AT Command Example Application*, corresponding to GR55xx SoC series.

### Revision History

Version	Date	Description
1.0	2021-02-23	Initial release
1.1	2021-04-20	Optimized descriptions in “Initial Operation” and “Application Details”.

# Contents

<b>Preface</b> .....	<b>I</b>
<b>1 Introduction</b> .....	<b>1</b>
<b>2 Profile Overview</b> .....	<b>2</b>
<b>3 Initial Operation</b> .....	<b>3</b>
3.1 Supported Development Platform.....	3
3.2 Firmware Download.....	3
3.3 Test and Verification.....	3
<b>4 Application Details</b> .....	<b>8</b>
4.1 Running Procedures.....	8
4.2 Major Code.....	10
4.2.1 Event Handler Function.....	10
4.2.2 Checking and Updating Environment Variables of AT Command.....	11
4.2.3 Performing Bluetooth LE Operations Specific to AT Command.....	12
4.2.4 Writing AT Command Execution Result to ble to uart Buffer.....	13
4.2.5 Reading Ring Buffers and Transmitting Data.....	14
<b>5 Custom Commands</b> .....	<b>16</b>
<b>6 FAQ</b> .....	<b>18</b>
6.1 Why do I Fail to Set GAP Roles Through AT Commands?.....	18
6.2 Why do I Fail to Set Device Information?.....	18
6.3 Why does “Invalid Input” Prompt Occur When Users Type an AT Command into GRUart?.....	18
<b>7 Appendix</b> .....	<b>19</b>
7.1 AT Command Table.....	19
7.2 Error Code.....	22

# 1 Introduction

GR55xx SDK provides an AT-command-related example, ble\_app\_uart\_at, to help developers quickly build a Bluetooth module and enable Bluetooth Low Energy (Bluetooth LE) communications. The example allows developers to control hardware through simple AT commands based on actual demands, freeing them up from modifying source code. AT commands feature easy extension and can be easily customized by users based on actual demands.

AT commands can be used to start/stop advertising, set advertising parameters, start/stop scanning, set scanning parameters, and get device name/address. In addition, the commands support control of devices equipped with SoCs through terminals. This makes integration of ble\_app\_uart\_at into third-party microcontrollers possible.

This document introduces how to use and verify the ble\_app\_uart\_at example in the GR55xx SDK.

Before you get started, it is recommended to refer to the following documents.

Table 1-1 Reference documents

Name	Description
Developer guide of the specific GR55xx SoC	Introduces the software/hardware and quick start guide of the specific GR55xx SoC in use.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at <a href="http://www.segger.com/downloads/jlink/UM08001_JLink.pdf">www.segger.com/downloads/jlink/UM08001_JLink.pdf</a> .
Keil User Guide	Offers detailed Keil operational instructions. Available at <a href="http://www.keil.com/support/man/docs/uv4/">www.keil.com/support/man/docs/uv4/</a> .

## 2 Profile Overview

Based on Goodix UART Service (GUS), the ble\_app\_uart\_at example is mainly used to enable passthrough. As the most easy-to-use means of Bluetooth LE communications, passthrough features:

- Unaltered service data during transmission
- Bidirectional data transfer

The GUS is identified by its vendor-specific Universally Unique Identifier (UUID), A6ED0201-D344-460A-8075-B9E8EC90D71B.

GUS includes three characteristics:

- GUS TX characteristic: Transmits data.
- GUS RX characteristic: Receives data.
- GUS Flow Control characteristic: Controls data flow.

These characteristics are described in detail as follows:

Table 2-1 GUS characteristics

Characteristic	UUID	Type	Support	Security	Property
GUS TX	A6ED0203-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Notify
GUS RX	A6ED0202-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write
GUS Flow Control	A6ED0204-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write/Notify

### 3 Initial Operation

This chapter introduces how to run and verify the ble\_app\_uart\_at example in the GR55xx SDK.

**Note:**

SDK\_Folder is the root directory for the GR55xx SDK in use.

#### 3.1 Supported Development Platform

You can use and modify the ble\_app\_uart\_at example on the following platform.

Table 3-1 Supported development platform

Hardware Platform	Development Board Model
GR551x development kit	GR5515-SK-BASIC

#### 3.2 Firmware Download

The source code of the ble\_app\_uart\_at example is in SDK\_Folder\projects\ble\ble\_multi\_role\ble\_app\_uart\_at.

You can download *ble\_app\_uart\_at\_fw.bin* to the GR55xx Starter Kit Board (GR55xx SK Board) through GProgrammer. For details, see *GProgrammer User Manual*.

For a project involving modification on source code of ble\_app\_uart\_at, re-compile the project to generate a new *ble\_app\_uart\_at\_fw.bin* file, and then download the file to the GR55xx SK Board. For details, refer to the developer guide of the specific GR55xx SoC in use.

**Note:**

- The *ble\_app\_uart\_at\_fw.bin* is in SDK\_Folder\projects\ble\ble\_multi\_role\ble\_uart\_at\build
- You can find GProgrammer in SDK\_Folder\tools\GProgrammer.

#### 3.3 Test and Verification

The hardware required for test and verification is listed in [Table 3-2](#).

Table 3-2 Hardware resource

Name	Description
GRUart (Windows)	A serial port debugging tool. Available in SDK_Folder\tools\GRUart.

In this test, two GR55xx SK Boards are used, with one named as GR55xx SK Board A (as Client) and the other as GR55xx SK Board B (as Server). The two boards communicate with each other through Bluetooth LE.

After the GR55xx SK Boards and GRUart are ready, start GRUart. Wait until GRUart displays device address information and **Goodix UART(AT) example start**. This indicates the ble\_app\_uart\_at firmware operates properly. The figure below shows the proper operation of the firmware on GR55xx SK Board B.

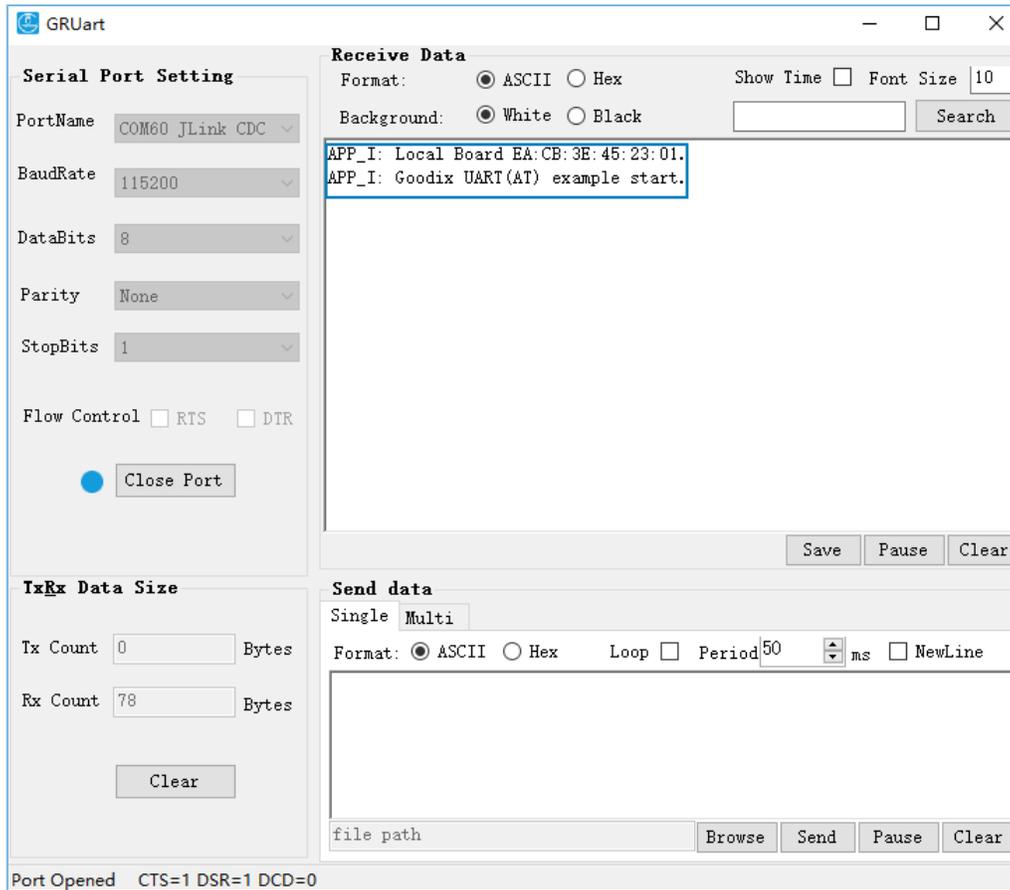


Figure 3-1 GRUart information displaying proper operation of firmware

**Note:**

The device address displayed on GRUart is the one generated after modifying source code of the ble\_app\_uart\_at example. The actual device address used by users prevails.

After the firmware of the ble\_app\_uart\_at example operates normally, deliver AT commands to perform specific Bluetooth LE operations.

1. Send AT:ADV\_STOP command to GR55xx SK Board B to stop advertising, after which send AT:ADV\_START command to restart advertising.

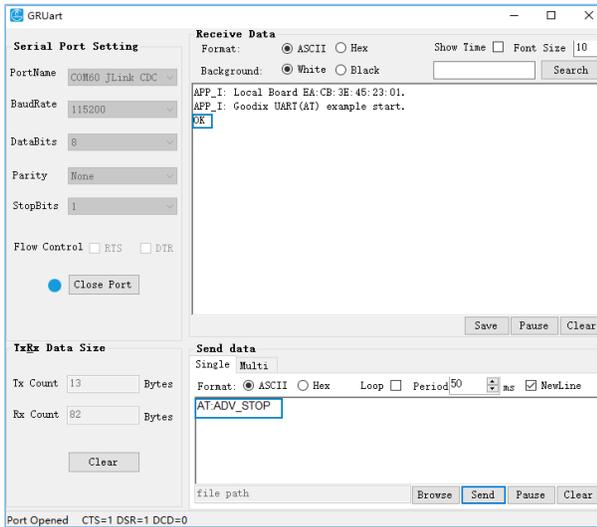


Figure 3-2 Stopping advertising

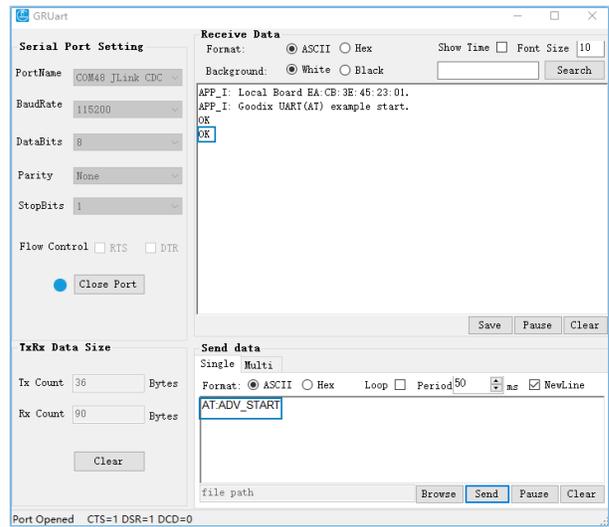


Figure 3-3 Restarting advertising

2. Send AT:SCAN\_START command to GR55xx SK Board A to start scanning. When GR55xx SK Board A discovers the GUS, send AT:CONN\_INIT= command to initiate a connection with GR55xx SK Board B.

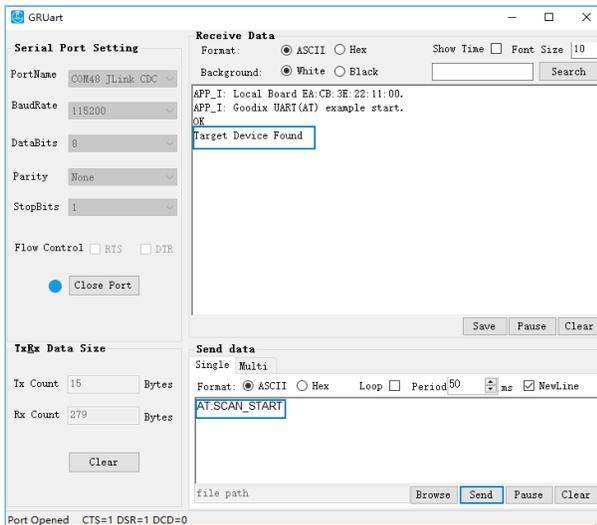


Figure 3-4 Starting scanning

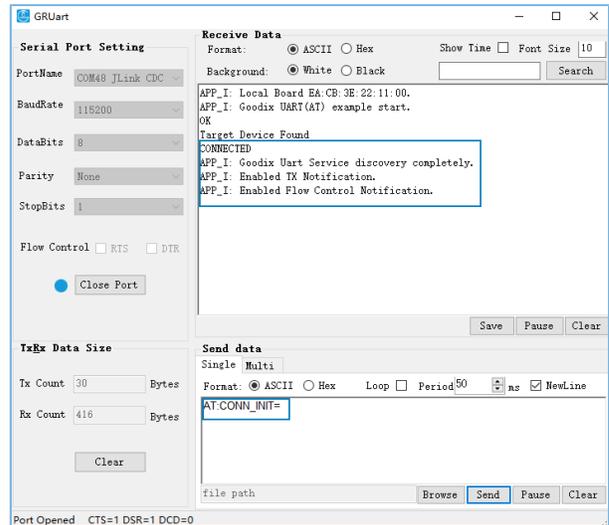


Figure 3-5 Initiating a connection after discovering GUS

3. After GR55xx SK Board A is successfully connected to GR55xx SK Board B, execute specific AT commands to get the address and role information of the two boards.
  - Send AT:ADDR? command to get address information.

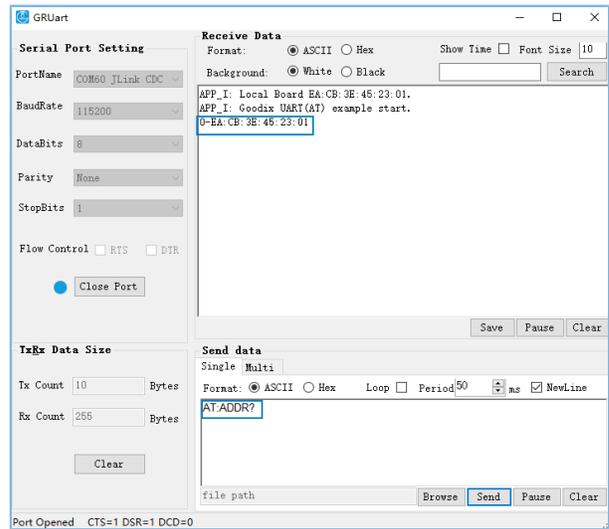
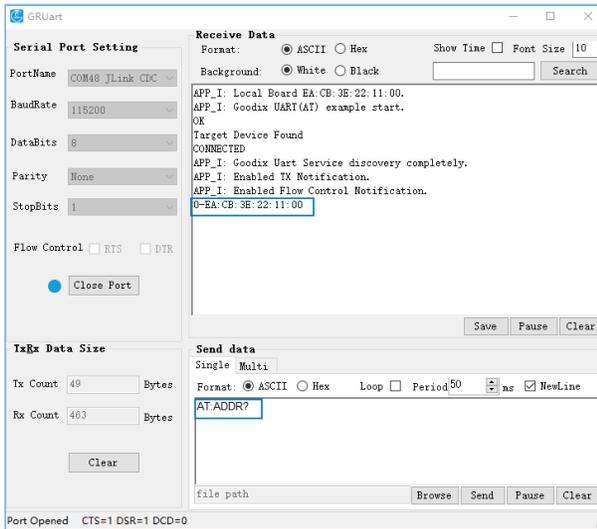


Figure 3-6 Getting device address information of GR55xx SK Board A      Figure 3-7 Getting device address information of GR55xx SK Board B

- Send AT:GAP\_ROLE? command to get role information.

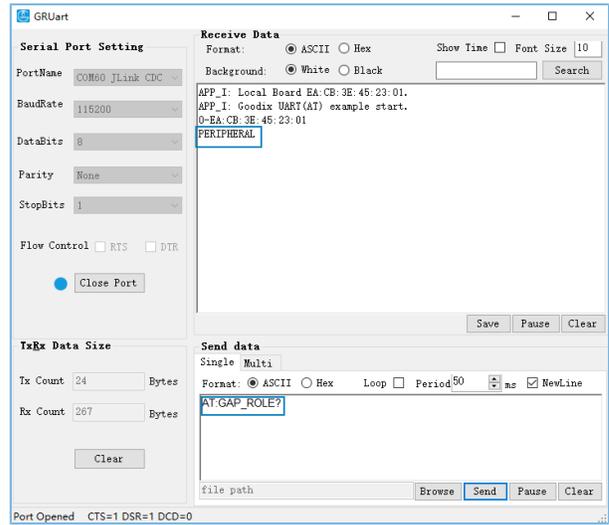
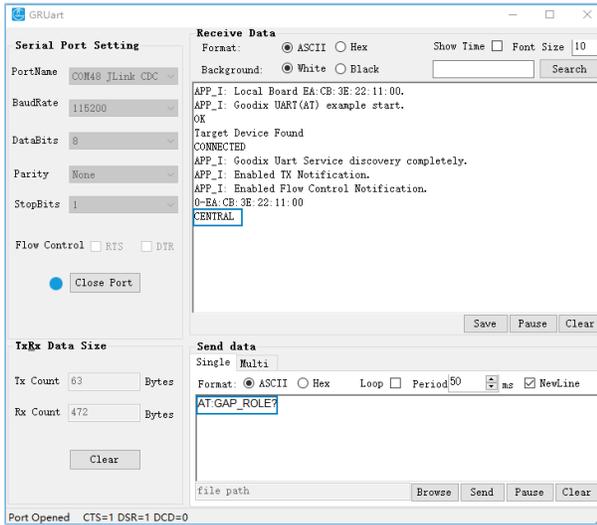


Figure 3-8 Getting role information of GR55xx SK Board A

Figure 3-9 Getting role information of GR55xx SK Board B

4. Enable data transmission via GUS.

- GR55xx SK Board B (Server) sends **Goodix\_BLE** to GR55xx SK Board A (Client).

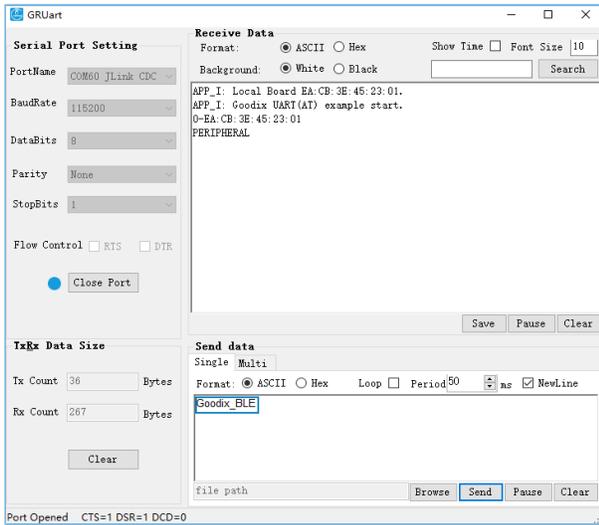


Figure 3-10 Server sends data to Client

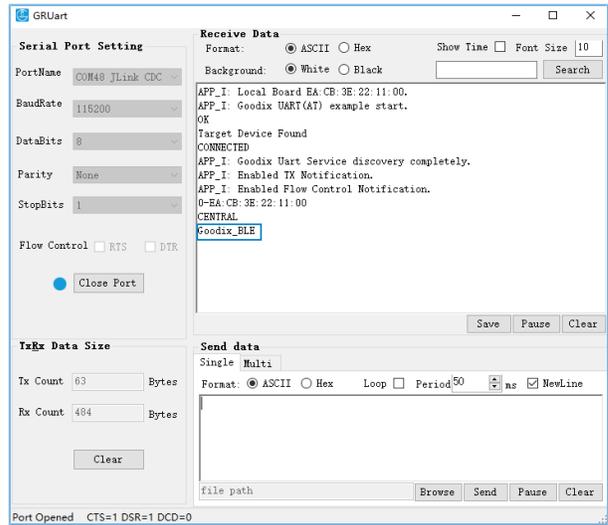


Figure 3-11 Client receives data from Server

- GR55xx SK Board A (Client) sends **Hello Word!** to GR55xx SK Board B (Server).

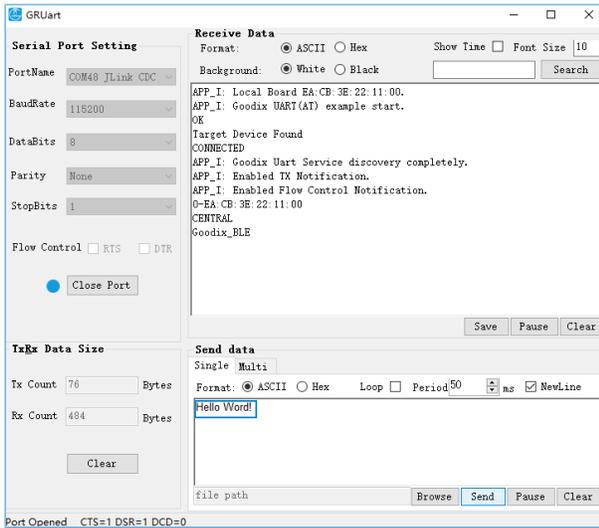


Figure 3-12 Client sends data to Server

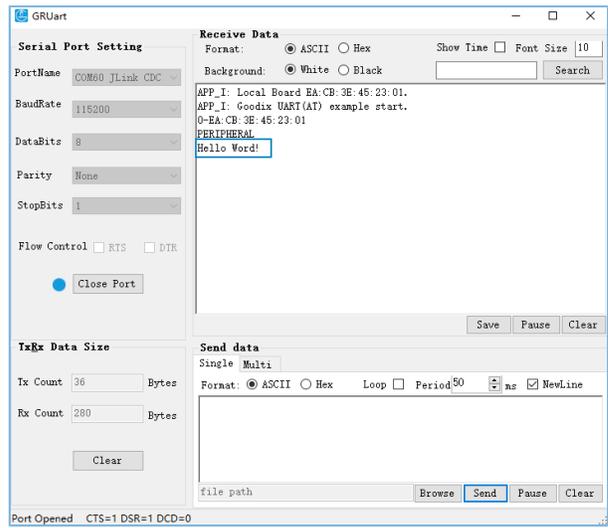


Figure 3-13 Server receives data from Client

## 4 Application Details

This chapter introduces the running procedures and major code of the ble\_app\_uart\_at example.

### 4.1 Running Procedures

This section elaborates on the running procedures of the ble\_app\_uart\_at example, aiming to help users deeply understand the operational mechanism of the example.

The following figure displays the running procedures of the ble\_app\_uart\_at example:

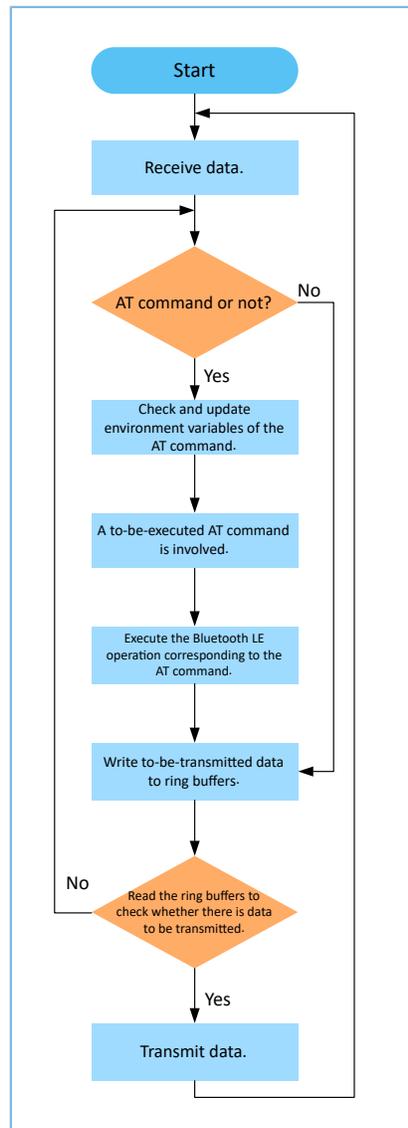


Figure 4-1 ble\_app\_uart\_at running procedures

1. Judge whether the received data is an AT command. If yes, check and update the environment variables of the AT command. If no, perform Step 4.
2. Read the environment variables of the AT command. When there is an AT command to be executed and its command handler is not null, perform Step 3.

3. Perform Bluetooth LE operations corresponding to the AT command.
4. Write to-be-transmitted data to ring buffers. The ring buffers comprise two types: ble to uart buffer (for storing received data) and uart to ble buffer (for storing to-be-transmitted data).
  - When AT commands are sent via GRUart, the command execution result is cached to the ble to uart buffer.
  - When non-AT commands are sent via GRUart, the data transmission mechanism is explained by taking two GR55xx SK Boards running ble\_app\_uart\_at firmware as an example. Connect the two GR55xx SK Boards to a PC and enable Bluetooth on the boards. GR55xx SK Board A (Client) sends non-AT commands to the GR55xx SK Board B (Server) via GRUart. The to-be-transmitted data will be cached into the uart to ble buffer before transmission by the Client, and the Server caches the received data to the ble to uart buffer.
5. Read the ring buffers to check whether there is data to be transmitted. If yes, transmit the data. Otherwise, return to Step 1.

The following figure displays how to execute an AT command.

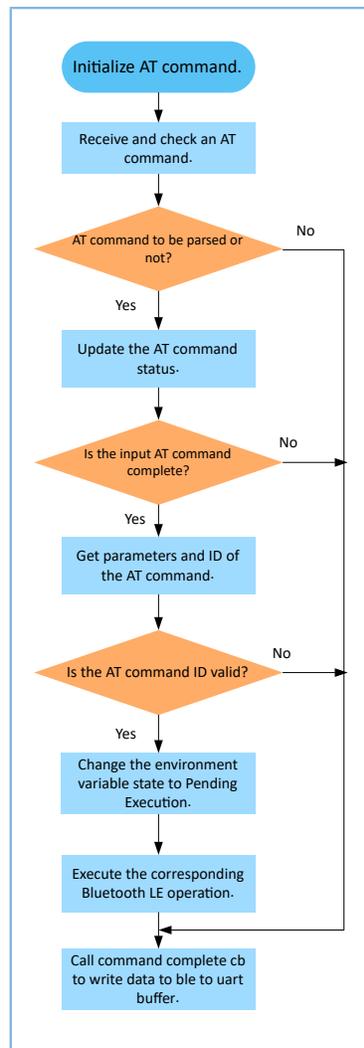


Figure 4-2 AT command execution procedures

1. Initialize AT command. Complete registration of the AT command attribute table, command complete cb, and app timer.

**Note:**

- AT command attribute table records AT command information such as AT Command ID, AT Command Tag, AT Command Tag Length, and AT Command Handler.
  - command complete cb writes the command execution result to the ble to uart buffer. If an error exists in the command response, the error code is regarded as to-be-transmitted data. If there is no error, the command response data is regarded as to-be-transmitted data.
  - app timer manages timeouts.
2. Check the received AT command. If the command is to be parsed, change the status of the command from Pending Parsing to In Parsing and perform Step 3. Otherwise, update the error code and perform Step 7.
  3. Check whether the input AT command is complete. A complete AT command starts with AT: and ends with \r\n. If the command is complete, perform Step 4. Otherwise, update the error code, and perform Step 7.
  4. Get parameters and ID of the AT command.
  5. Check whether the AT command ID is valid. If the command ID is valid, change the status of the command to Pending Execution. Otherwise, update the error code, and perform Step 7.
  6. Execute the AT command. If the AT Command Handler is not null, call the handler to perform related Bluetooth LE operation. Otherwise, update the error code, and perform Step 7.
  7. Call command complete cb to write the command execution result to the ble to uart buffer.

## 4.2 Major Code

This section introduces the major code of the ble\_app\_uart\_at example.

### 4.2.1 Event Handler Function

**Path:** user\_app\user\_app.c

**Name:** gus\_service\_process\_event();

When a data receiving event occurs, the event handler function checks whether the received data is an AT command. If yes, call at\_cmd\_parse. If no, write the data to ble to uart buffer.

```
void at_cmd_schedule(void)
{
    uint8_t ble_rx_data[AT_CMD_BUFFER_SIZE_MAX];
    switch (p_evt->evt_type)
    {
        ...
        case GUS_EVT_RX_DATA_RECEIVED:
            if (0 == memcmp(p_evt->p_data, "AT:" , 3))
            {
                memcpy(ble_rx_data, p_evt->p_data, p_evt->length);
            }
            break;
    }
}
```

```

        if ((0x0d != p_evt->p_data[p_evt->length - 2]) ||\
            (0x0a != p_evt->p_data[p_evt->length - 1]))
        {
            ble_rx_data[p_evt->length]      = 0x0d;
            ble_rx_data[p_evt->length + 1] = 0x0a;
        }

        at_cmd_parse(AT_CMD_SRC_BLE, ble_rx_data, p_evt->length + 2);
    }
    else
    {
        ble_to_uart_buff_data_push(p_evt->p_data, p_evt->length);
    }

    break;
    ...
}
}

```

## 4.2.2 Checking and Updating Environment Variables of AT Command

**Path:** gr\_libraries\at\_cmd.c under the project directory

**Name:** at\_cmd\_parse();

If the received data is an AT command, check and update the environment variables of the command. If the command passes all checks, set the environment variable state of the command to Pending Execution (AT\_CMD\_IN\_WAITE\_EXECUTE). The code snippet is as follows:

```

void at_cmd_parse(at_cmd_src_t cmd_src, const uint8_t *p_data, uint16_t length)
{
    AT_CMD_RSP_DEF(cmd_rsp);

    s_at_cmd_env.cmd_src = cmd_src;

    // Check parse cmd is allowed or not
    if (AT_CMD_IN_READY_PARSE != s_at_cmd_env.cmd_state)
    {
        cmd_rsp.error_code = AT_CMD_ERR_PARSE_NOT_ALLOWED;
        at_cmd_execute_cplt(&cmd_rsp);
        return;
    }
    else
    {
        s_at_cmd_env.cmd_state = AT_CMD_IN_PARSING;
    }

    // Check cmd input is integrity or not
    if (!at_cmd_integrity_check(p_data, length, &s_parse_rlt))
    {
        cmd_rsp.error_code = AT_CMD_ERR_INVALID_INPUT;
        at_cmd_execute_cplt(&cmd_rsp);
        return;
    }

    // Get cmd parameters
    at_cmd_args_get(&s_parse_rlt);

    // Get cmd Id
    at_cmd_id_get(&s_parse_rlt);
}

```

```

// Check cmd id is valid or not
if (AT_CMD_INVALID == s_parse_rlt.cmd_id)
{
    cmd_rsp.error_code = AT_CMD_ERR_UNSUPPORTED_CMD;
    at_cmd_execute_cplt(&cmd_rsp);
    return;
}

s_at_cmd_env.cmd_state = AT_CMD_IN_WAITE_EXECUTE;
}

```

### 4.2.3 Performing Bluetooth LE Operations Specific to AT Command

**Path:** gr\_libraries\at\_cmd.c under the project directory

**Name:** at\_cmd\_schedule();

Read the environment variables of the AT command. If the command is in Pending Execution state with the Command Handler being not null, perform Bluetooth LE operations such as advertising, scanning, and connection based on the attribute table of the AT command. If the AT command is used to modify device name, call the command handler function, uart\_at\_gap\_name\_set, to modify the device name.

```

void at_cmd_schedule(void)
{
    if (AT_CMD_IN_WAITE_EXECUTE == s_at_cmd_env.cmd_state)
    {
        s_at_cmd_env.cmd_state = AT_CMD_IN_EXECUTING;

        if (s_at_cmd_env.p_cmd_attr[s_parse_rlt.cmd_idx].cmd_handler)
        {
            if (s_at_cmd_env.cmd_time_cb)
            {
                s_at_cmd_env.cmd_time_cb();
            }

            s_at_cmd_env.p_cmd_attr[s_parse_rlt.cmd_idx].cmd_handler(&s_parse_rlt);
        }
        else
        {
            AT_CMD_RSP_DEF(cmd_rsp);
            cmd_rsp.error_code = AT_CMD_ERR_NO_CMD_HANDLER;
            at_cmd_execute_cplt(&cmd_rsp);
        }
    }
}

```

**Path:** user\_app\at\_cmd\_handler.c under the project directory

**Name:** uart\_at\_gap\_name\_set();

The code snippet is as follows:

```

void uart_at_gap_name_set(at_cmd_parse_t *p_cmd_param)
{
    AT_CMD_RSP_DEF(cmd_rsp);
    sdk_err_t    error_code;
    uint32_t     index;

    if (2 != p_cmd_param->arg_count)

```

```

{
    cmd_rsp.error_code = AT_CMD_ERR_INVALID_PARAM;
}
else
{
    if (at_cmd_decimal_num_check(&p_cmd_param->p_buff[p_cmd_param->arg_idx[0]],
                                p_cmd_param->arg_length[0], &index))
    {
        error_code = ble_gap_device_name_set((gap_dev_name_write_perm_t)index,
        &p_cmd_param->p_buff[p_cmd_param->arg_idx[1]], p_cmd_param-
>arg_length[1]);

        cmd_rsp.error_code = at_cmd_ble_err_convert(error_code);
    }
    else
    {
        cmd_rsp.error_code = AT_CMD_ERR_INVALID_PARAM;
    }
}

if (AT_CMD_ERR_NO_ERROR == cmd_rsp.error_code)
{
    cmd_rsp.length = at_cmd_printf_bush(cmd_rsp.data, "OK");
}

at_cmd_execute_cplt(&cmd_rsp);
}

```

#### 4.2.4 Writing AT Command Execution Result to ble to uart Buffer

**Path:** gr\_libraries\at\_cmd.c under the project directory

**Name:** at\_cmd\_execute\_cplt();

After the AT Command Handler is executed successfully, update the to-be-transmitted data based on the return value of the AT command. Call cmd\_cplt\_cb to write the execution result of the command to the ble to uart buffer. The code snippet is as follows:

```

void at_cmd_execute_cplt(at_cmd_rsp_t *p_cmd_rsp)
{
    uint8_t length = 0;

    if (AT_CMD_ERR_NO_ERROR != p_cmd_rsp->error_code)
    {
        switch(p_cmd_rsp->error_code)
        {
            ...
            case AT_CMD_ERR_UNSUPPORTED_CMD:
                length = at_cmd_printf_bush(at_cmd_rsp_buff, "ERR: Unsupported AT CMD.");
                break;
            ...
        }
    }
    else
    {
        memcpy(at_cmd_rsp_buff, p_cmd_rsp->data, p_cmd_rsp->length);
        length = p_cmd_rsp->length;
    }

    at_cmd_rsp_buff[length] = 0x0d;
}

```

```

at_cmd_rsp_buff[length + 1] = 0x0a;

if (s_at_cmd_env.cmd_cplt_cb)
{
    if (AT_CMD_SRC_UART == s_at_cmd_env.cmd_src)
    {
        s_at_cmd_env.cmd_cplt_cb(AT_CMD_RSP_DEST_UART, at_cmd_rsp_buff, length + 2);
    }
    else if (AT_CMD_SRC_BLE == s_at_cmd_env.cmd_src)
    {
        s_at_cmd_env.cmd_cplt_cb(AT_CMD_RSP_DEST_BLE, at_cmd_rsp_buff, length + 2);
    }
}

s_at_cmd_env.cmd_state = AT_CMD_IN_READY_PARSE;
...
}

```

**Path:** user\_app\at\_cmd\_handler.c under the project directory

**Name:** user\_at\_cmd\_callback();

```

static void user_at_cmd_callback(at_cmd_rsp_dest_t rsp_dest, const uint8_t *p_data,
                                uint8_t length)
{
    s_curr_rsp_dest = rsp_dest;

    if (AT_CMD_RSP_DEST_UART == s_curr_rsp_dest)
    {
        ble_to_uart_buff_data_push(p_data, length);
    }
    else if (AT_CMD_RSP_DEST_BLE == s_curr_rsp_dest)
    {
        uart_to_ble_buff_data_push(p_data, length);
    }

    app_timer_delete(&s_at_cmd_timing_id);
}

```

## 4.2.5 Reading Ring Buffers and Transmitting Data

**Path:** user\_app\transport\_scheduler.c under the project directory

**Name:** transport\_schedule();

When Notify and Flow Control characteristics on the device are enabled, if data exists in the ring buffers, read the data; then transmit the read data.

```

void transport_schedule(void)
{
    uint16_t items_avail    = 0;
    uint16_t read_len      = 0;

    // read data from s_uart_to_ble_buffer, then notify or write to peer.
    if (transport_flag_cfm(GUS_TX_NTF_ENABLE) && transport_flag_cfm(BLE_TX_CPLT) &&
        transport_flag_cfm(BLE_TX_FLOW_ON))
    {
        items_avail = ring_buffer_items_count_get(&s_uart_to_ble_buffer);
    }
}

```

```
if (items_avail > 0)
{
    read_len = ring_buffer_read(&s_uart_to_ble_buffer, s_ble_tx_data,
                                s_mtu_size - 3);

    transport_flag_set(BLE_TX_CPLT, false);

    if (BLE_GAP_ROLE_PERIPHERAL == uart_at_curr_gap_role_get())
    {
        gus_tx_data_send(0, s_ble_tx_data, read_len);
    }
    else if (BLE_GAP_ROLE_CENTRAL == uart_at_curr_gap_role_get())
    {
        gus_c_tx_data_send(0, s_ble_tx_data, read_len);
    }
}

// read data from s_ble_to_uart_buffer, then send to uart.
items_avail = ring_buffer_items_count_get(&s_ble_to_uart_buffer);

if (items_avail > 0)
{
    read_len = ring_buffer_read(&s_ble_to_uart_buffer, s_uart_tx_data,
                                ONCE_SEND_DATA_SIZE);
    uart_tx_data_send(s_uart_tx_data, read_len);
}
}
```

## 5 Custom Commands

This chapter depicts how to customize AT commands when using and verifying the `ble_app_uart_at` example.

Add necessary elements of custom commands into the AT command attribute table. The elements include AT Command ID, AT Command, AT Command Length, and AT Command Handler. After that, implement the Command Handler(s).

### Note:

You can find the AT command attribute table in `user_app\at_cmd_handler.c` under the project directory.

Take the custom AT command for MTU exchange as an example. Follow the steps below to add the command to the attribute table.

1. Add the required AT Command ID to the `at_cmd_id_t` structure in the `at_cmd.h` file (available in `SDK_Folder\components\libraries\at_cmd`).
2. Update the AT command attribute table in code, and add necessary command elements to `s_at_cmd_attr_table`.

The updated AT command attribute table is displayed below:

```
static at_cmd_attr_t s_at_cmd_attr_table[] =
{
    {AT_CMD_INVALID,          "",          0, NULL},
    {AT_CMD_TEST,            "TEST",    4, uart_at_test},
    {AT_CMD_VERSION_GET,     "VERSION?", 8, uart_at_version_get},
    {AT_CMD_RESET,          "RESET",   5, uart_at_app_reset},
    {AT_CMD_BAUD_SET,       "BAUD=",  5, uart_at_baud_set},
    {AT_CMD_ADDR_GET,       "ADDR?",  5, uart_at_bd_addr_get},
    {AT_CMD_GAP_ROLE_GET,   "GAP_ROLE?", 9, uart_at_gap_role_get},
    {AT_CMD_GAP_ROLE_SET,   "GAP_ROLE=", 9, uart_at_gap_role_set},
    {AT_CMD_GAP_NAME_GET,   "GAP_NAME?", 9, uart_at_gap_name_get},
    {AT_CMD_GAP_NAME_SET,   "GAP_NAME=", 9, uart_at_gap_name_set},
    {AT_CMD_ADV_PARAM_SET,  "ADV_PARAM=", 10, uart_at_adv_param_set},
    {AT_CMD_ADV_START,      "ADV_START", 9, uart_at_adv_start},
    {AT_CMD_ADV_STOP,       "ADV_STOP", 8, uart_at_adv_stop},
    {AT_CMD_SCAN_PARAM_SET, "SCAN_PARAM=", 11, uart_at_scan_param_set},
    {AT_CMD_SCAN_START,     "SCAN_START", 10, uart_at_scan_start},
    {AT_CMD_SCAN_STOP,      "SCAN_STOP", 9, uart_at_scan_stop},
    {AT_CMD_CONN_PARAM_SET, "CONN_PARAM=", 11, uart_at_conn_param_set},
    {AT_CMD_CONN_INIT,      "CONN_INIT=", 10, uart_at_conn_init},
    {AT_CMD_CONN_CANCEL,    "CONN_CANCEL", 11, uart_at_conn_cancel},
    {AT_CMD_DISCONN,        "DISCONN", 7, uart_at_disconnect},
    {AT_CMD_MTU_EXCHANGE,   "MTU_EXC", 7, uart_at_mtu_exchange},
};
```

### Note:

Added code is in bold.

3. Implement AT Command Handler.

```
void uart_at_mtu_exchange(at_cmd_parse_t *p_cmd_param)
{
    AT_CMD_RSP_DEF(cmd_rsp);
    sdk_err_t error_code;
```

```
error_code = ble_gattc_mtu_exchange(0);
cmd_rsp.error_code = at_cmd_ble_err_convert(error_code);

if (AT_CMD_ERR_NO_ERROR != cmd_rsp.error_code)
{
    at_cmd_execute_cplt(&cmd_rsp);
}
}
```

## 6 FAQ

### 6.1 Why do I Fail to Set GAP Roles Through AT Commands?

- Description  
Users cannot set GAP roles by using AT commands.
- Analysis  
The device is not in standby state, resulting in failure to set GAP roles through AT commands.
- Solution  
Ensure the device is in standby state when using AT commands to set GAP roles.

### 6.2 Why do I Fail to Set Device Information?

- Description  
Users cannot set device information by using AT commands.
- Analysis  
When setting device information such as modifying GAP roles and names, a space exists after "=" in an AT command.
- Solution  
Make sure there is no space after "=" in an AT command.

### 6.3 Why does “Invalid Input” Prompt Occur When Users Type an AT Command into GRUart?

- Description  
GRUart prompts that the input AT command is invalid.
- Analysis  
An AT command shall end with `\r\n`. When users type an AT command on GRUart, **NewLine** in the **Single** tab under **Send data** is unchecked.
- Solution  
Remember to check **NewLine** in the **Single** tab under **Send data** on GRUart.

## 7 Appendix

### 7.1 AT Command Table

The table below shows the AT commands involved in the ble\_app\_uart\_at example.

Table 7-1 Supported AT commands for ble\_app\_uart\_at

Command Type	AT Command	Description	Return Value	Example
Test	AT:TEST	Tests whether AT command operates properly.	OK	AT:TEST
Version	AT:VERSION?	Gets the device version number.	Version number	AT:VERSION?
System reset	AT:RESET	Resets system.	-	AT:RESET
Baud rate	AT:BAUD= <NEW_VALUE>	Configures baud rate. NEW_VALUE: baud rate; range: [0, 2000000]	Successful: OK Failed: ERR: Invalid parameters.	AT:BAUD=4900
Device address	AT:ADDR?	Gets device address.	Successful: Device address Failed: No device information is returned.	AT:ADDR?
GAP role	AT:GAP_ROLE?	Gets role information of the device.	Device roles including NONE, OBSERVER, BROADCASTER, CENTRAL, PERIPHERAL, ALL	AT:GAP_ROLE?
	AT:GAP_ROLE= <NEW_ROLE>	Sets device role. NEW_ROLE: device role. Options include N, n, O, o, B, b, C, c, P, p, A, and a.	Successful: OK Failed: ERR: Command request is not allowed.	AT:GAP_ROLE=0
GAP name	AT:GAP_NAME?	Gets device name.	Successful: Device name Failed: Specific error code	AT:GAP_NAME?
	AT:GAP_NAME=<INDEX, NEW_NAME>	Sets the device name. INDEX: Write permission of the device name; options: 0, 1, 2, 3, and 4	Successful: OK Failed: Specific error code	AT:GAP_NAME=1,Goodix

Command Type	AT Command	Description	Return Value	Example
		<ul style="list-style-type: none"> <li>0: Write not allowed</li> <li>1: Link neither encrypted nor authenticated</li> <li>2: Link encrypted but not authenticated</li> <li>3: Link encrypted and authenticated (MITM)</li> <li>4: Link encrypted and authenticated (secure connections)</li> </ul> NEW_NAME: Custom advertising name		
Advertise	AT:ADV_PARAM=<ADV_INTERVAL, ADV_DURATION>	Sets advertising parameters. <ul style="list-style-type: none"> <li>ADV_INTERVAL: Advertising interval; unit: 0.625 ms; range: &gt; 32.</li> <li>ADV_DURATION: Advertising duration; unit: 10 ms;</li> </ul> When ADV_DURATION = 0, the device will continue advertising until the host disables it. For Limited Discoverable Mode, the parameter value ranges from 1 to 18000; for directed advertising with high duty cycle, the parameter value ranges from 1 to 128.	Successful: OK Failed: Specific error code	AT:ADV_PARAM=80,0
	AT:ADV_START	Starts advertising.	Successful: OK Failed: Specific error code	AT:ADV_START
	AT:ADV_STOP	Stops advertising.	Successful: OK	AT:ADV_STOP

Command Type	AT Command	Description	Return Value	Example
			Failed: Specific error code	
Scan	AT:SCAN_PARAM=<SCAN_INTERVAL,SCAN_DURATION>	<p>Sets scanning parameters.</p> <ul style="list-style-type: none"> <li>SCAN_INTERVAL: Scanning interval; unit: 0.625 ms; range: 4 to 16384</li> <li>SCAN_DURATION: Scanning duration; unit: 0.625 ms; range: 1 to 65535</li> </ul>	<p>Successful: OK</p> <p>Failed: Specific error code</p>	AT:SCAN_PARAM=176,1000
	AT:SCAN_START	Starts scanning.	<p>Successful: OK</p> <p>Failed: Specific error code</p>	AT:SCAN_START
	AT:SCAN_STOP	Stops scanning.	<p>Successful: OK</p> <p>Failed: Specific error code</p>	AT:SCAN_STOP
Connect	AT:CONN_PARAM=<CONN_INTERVAL,CONN_LATENCY,CONN_SUP_TIMEOUT>	<p>Sets connection parameters.</p> <ul style="list-style-type: none"> <li>CONN_INTERVAL: Connection interval; range: 6 to 3200</li> <li>CONN_LATENCY: Number of connection events that can be ignored; range: &lt;math&gt;(\text{CONN\_SUP\_TIMEOUT} / \text{CONN\_INTERVAL}) - 1&lt;/math&gt;</li> <li>CONN_SUP_TIMEOUT: Supervision timeout; range: 10 to 3200</li> </ul>	<p>Successful: OK</p> <p>Failed: Specific error code</p>	AT:CONN_PARAM=12,5,3200
	AT:CONN_INIT=	Initiates a connection.	<p>Successful: CONNECTED</p> <p>Failed: Specific error code</p>	AT:CONN_INIT=
	AT:CONN_CANCEL	Terminates a connection.	<p>Successful: OK</p> <p>Failed: Specific error code</p>	AT:CONN_CANCEL
Disconnect	AT:DISCONN	Disconnects a connection.	<p>Successful: DISCONNECTED</p> <p>Failed: Specific error code</p>	AT:DISCONN
MTU exchange	AT:MTU_EXC	Exchanges an MTU.	<p>Successful: MTU</p> <p>Failed: Specific error code</p>	AT:MTU_EXC

## 7.2 Error Code

When a failure occurs during executing AT commands, error code will be returned. The table below lists error code involved when using and verifying the ble\_app\_uart\_at example.

Table 7-2 Error code definitions

Name	Description
AT_CMD_ERR_INVALID_INPUT	The input information is invalid.
AT_CMD_ERR_UNSUPPORTED_CMD	The input AT command is not supported.
AT_CMD_ERR_PARSE_NOT_ALLOWED	The AT command cannot be parsed.
AT_CMD_ERR_CMD_REQ_ALLOWED	The command request is not allowed; for instance, if the device is not in standby state when users set GAP roles through AT commands, AT_CMD_ERR_CMD_REQ_ALLOWED is returned.
AT_CMD_ERR_NO_CMD_HANDLER	The AT Command Handler is null.
AT_CMD_ERR_INVALID_PARAM	The input AT command parameter is invalid.
AT_CMD_ERR_HAL_ERROR	A timeout occurs for HAL operations.
AT_CMD_ERR_TIMEOUT	A timeout occurs when the AT command is executed.
AT_CMD_ERR_OTHER_ERROR	Other errors