



## GR551x AT Command示例手册

版本： 1.3

发布日期： 2022-02-20

版权所有 © 2022 深圳市汇顶科技股份有限公司。保留一切权利。

非经本公司书面许可，任何单位和个人不得对本手册内的任何部分擅自摘抄、复制、修改、翻译、传播，或将其全部或部分用于商业用途。

## 商标声明

**GOODIX** 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

## 免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田区腾飞工业大厦B座2层、13层

电话：+86-755-33338828      传真：+86-755-33338099

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

# 前言

## 编写目的

本文档介绍了如何使用和验证GR551x SDK中的ble\_app\_uart\_at示例，旨在帮助用户快速进行二次开发。

## 读者对象

本文适用于以下读者：

- GR551x用户
- GR551x开发人员
- GR551x测试人员
- 开发爱好者

## 版本说明

本文档为第4次发布，对应的产品系列为GR551x。

## 修订记录

版本	日期	修订内容
1.0	2021-02-23	首次发布
1.1	2021-04-20	优化“初次运行”和“应用详解”章节
1.2	2021-08-09	更新“准备工作”章节
1.3	2022-02-20	基于SDK修改固件名称

# 目录

前言.....	I
1 简介.....	1
2 Profile概述.....	2
3 初次运行.....	3
3.1 准备工作.....	3
3.2 固件烧录.....	3
3.3 测试验证.....	4
4 应用详解.....	8
4.1 运行流程.....	8
4.2 关键代码.....	10
4.2.1 事件处理函数.....	10
4.2.2 校验更新AT Command环境变量.....	11
4.2.3 执行AT Command对应的Bluetooth LE操作.....	12
4.2.4 将AT Command执行结果写入到ble to uart buffer中.....	13
4.2.5 读取Ring Buffer并发送数据.....	15
5 自定义命令.....	17
6 常见问题.....	19
6.1 设置GAP角色失败.....	19
6.2 设置设备信息失败.....	19
6.3 使用GRUart传输AT Command时，提示输入无效.....	19
7 附录.....	20
7.1 AT指令表.....	20
7.2 Error Code详解.....	22

## 1 简介

为快速建立蓝牙模组，实现Bluetooth Low Energy（Bluetooth LE）通信，GR551x SDK提供了ble\_app\_uart\_at示例。该示例使得开发者可以通过简单的AT Command实现对硬件设备的控制，无需编写任何源代码。AT Command易于扩展，可根据实际需求自定义。

AT Command可用于开启/停止广播、设置广播参数、开启/停止扫描、设置扫描参数、获取设备名称和地址等，也可从终端直接对设备进行控制，因而ble\_app\_uart\_at示例可以集成到第三方的微控制器中。

本文将介绍如何使用和验证GR551x SDK中的ble\_app\_uart\_at示例。

在进行操作前，可参考以下文档。

表 1-1 文档参考

名称	描述
GR551x开发者指南	介绍GR551x SDK以及基于SDK的应用开发和调试
J-Link用户指南	J-Link使用说明: <a href="http://www.segger.com/downloads/jlink/UM08001_JLink.pdf">www.segger.com/downloads/jlink/UM08001_JLink.pdf</a>
Keil用户指南	Keil详细操作说明: <a href="http://www.keil.com/support/man/docs/uv4/">www.keil.com/support/man/docs/uv4/</a>

## 2 Profile概述

ble\_app\_uart\_at示例基于Goodix自定义的Goodix UART Service（GUS），主要用于实现透传功能。透传是Bluetooth LE中最简单的通信方式，具有以下特点：

- 对传输的业务数据内容不做任何改变。
- 双向传输。

GUS专用128位Universally Unique Identifier（UUID）为A6ED0201-D344-460A-8075-B9E8EC90D71B。

GUS包含三个特征：

- GUS TX Characteristic：发送数据。
- GUS RX Characteristic：接收数据。
- GUS Flow Control Characteristic：对数据流量进行控制。


Characteristic的具体描述如下表所示：

表 2-1 GUS Characteristic

Characteristic	UUID	Type	Support	Security	Properties
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 初次运行

本章主要介绍如何运行和验证GR551x SDK中的ble\_app\_uart\_at示例。

 说明:

SDK\_Folder为GR551x SDK的根目录。

### 3.1 准备工作

运行ble\_app\_uart\_at示例之前，请完成以下准备工作。

- 硬件准备

表 3-1 硬件准备

名称	描述
开发板	GR5515 Starter Kit开发板（以下简称“开发板”）2块
数据线	Micro USB 2.0数据线

- 软件准备

表 3-2 软件准备

名称	描述
Windows	Windows 7/Windows 10操作系统
J-Link Driver	J-Link驱动程序，下载网址： <a href="http://www.segger.com/downloads/jlink/">www.segger.com/downloads/jlink/</a>
Keil MDK5	IDE工具，支持MDK-ARM 5.20 及以上版本，下载网址： <a href="http://www.keil.com/download/product/">www.keil.com/download/product/</a>
GProgrammer（Windows）	Programming工具，位于SDK_Folder\tools\GProgrammer
GRUart（Windows）	串口调试工具，位于SDK_Folder\tools\GRUart

### 3.2 固件烧录

AT Command示例工程的源码位于SDK\_Folder\projects\ble\ble\_multi\_role\ble\_app\_uart\_at。

用户可通过GProgrammer将ble\_app\_uart\_at示例的ble\_app\_uart\_at.bin固件分别烧录至两块开发板。GProgrammer烧录固件的具体操作方法，请参考《GProgrammer用户手册》。

如果修改了ble\_app\_uart\_at示例工程，需重新编译示例工程后将生成的ble\_app\_uart\_at.bin固件下载至开发板。编译示例工程的具体操作请参考《GR551x开发者指南》。

 说明:

ble\_app\_uart\_at.bin位于SDK\_Folder\projects\ble\ble\_multi\_role\ble\_uart\_at\build。

### 3.3 测试验证

验证AT Command示例需使用两块开发板：开发板A和开发板B，分别作为Client端设备和Server端设备。两块开发板之间通过Bluetooth LE无线连接。

开发板、串口工具启动后，当GRUart界面出现设备地址信息以及“Goodix UART(AT) example start”提示信息时，表明ble\_app\_uart\_at固件运行正常。下图为开发板B中固件正常运行的示例图。

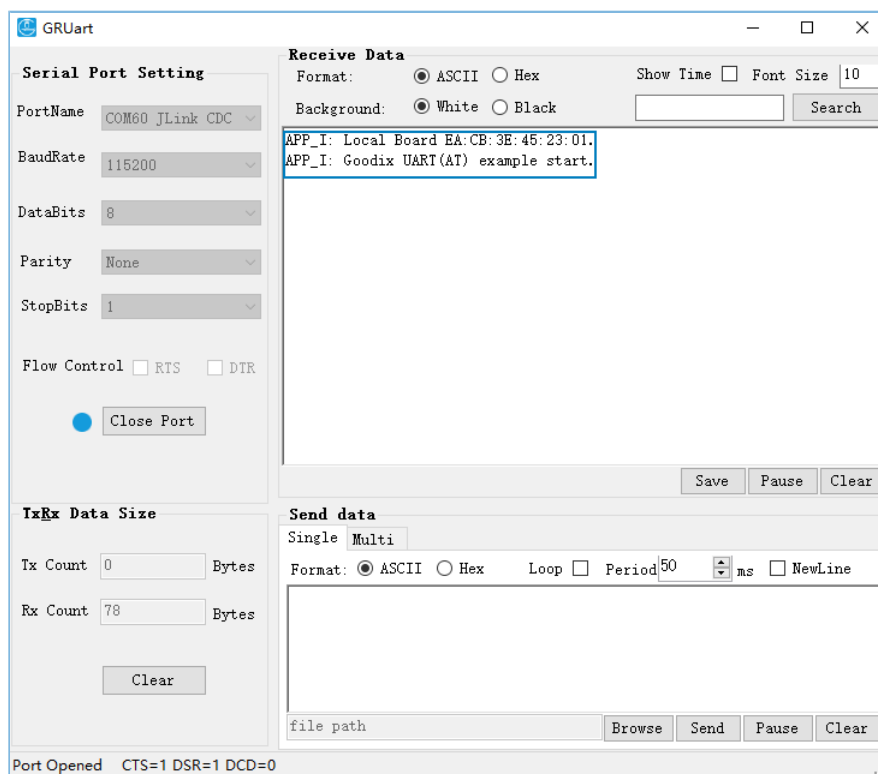


图 3-1 固件正常运行

#### 说明:

本文串口工具界面显示的设备地址为修改ble\_app\_uart\_at原示例后的地址，具体以用户实际的地址为准。

ble\_app\_uart\_at固件正常运行之后即可使用不同的AT Command进行对应的Bluetooth LE操作。

1. 对开发板B发送AT:ADV\_STOP命令停止广播，然后发送AT:ADV\_START命令重新开启广播。



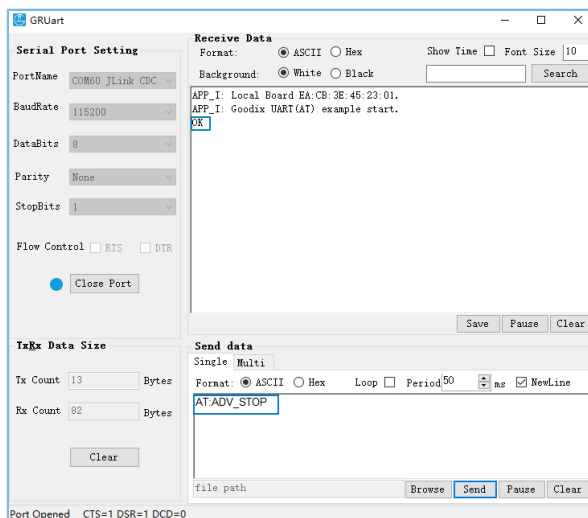


图 3-2 停止广播

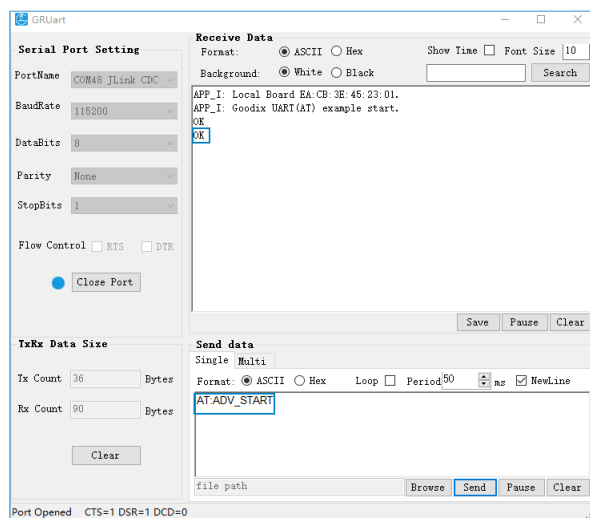


图 3-3 开启广播

2. 对开发板A使用AT:SCAN\_START命令开启扫描广播，发现GUS服务后使用AT:CONN\_INIT=命令向开发板B发起连接。

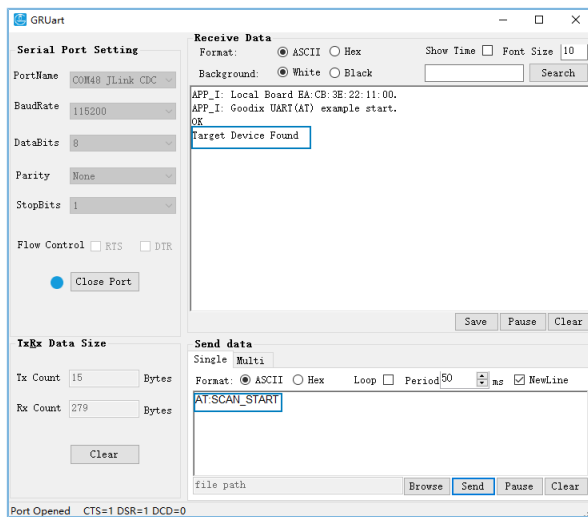


图 3-4 开启扫描广播

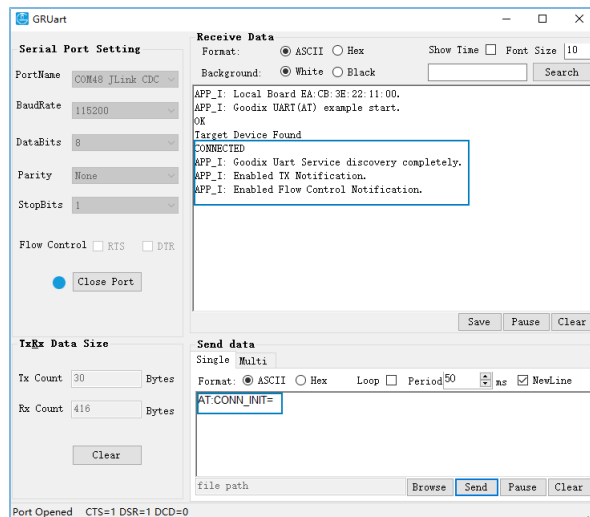


图 3-5 发现服务后发起连接

3. 两块开发板建立连接后，使用AT Command分别获取两块开发板的地址及角色信息。
  - 使用AT:ADDR?命令获取地址信息

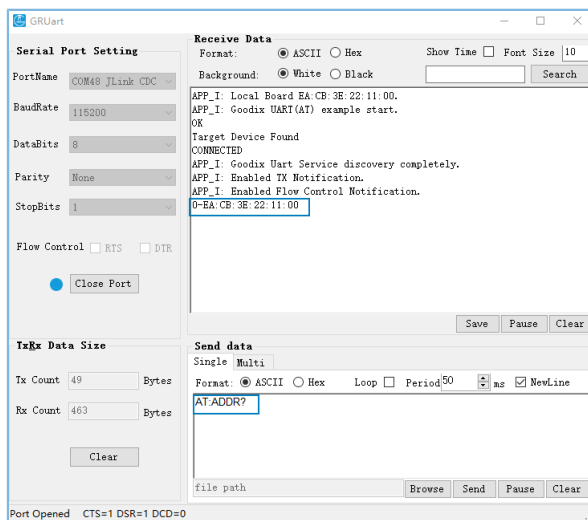


图 3-6 获取开发板A地址信息

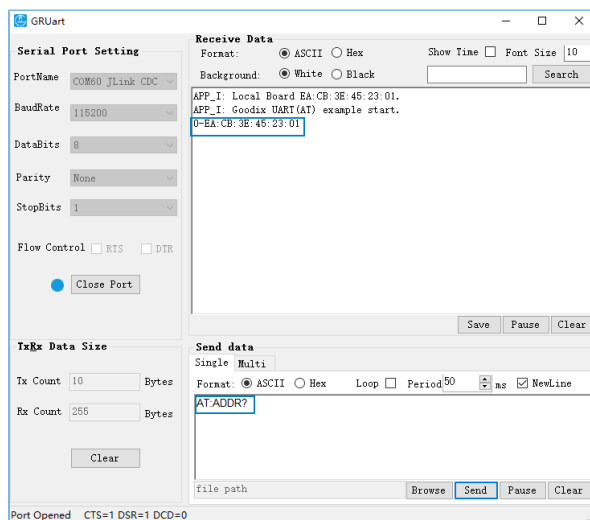


图 3-7 获取开发板B地址信息

- 使用AT:GAP\_ROLE?命令获取角色信息

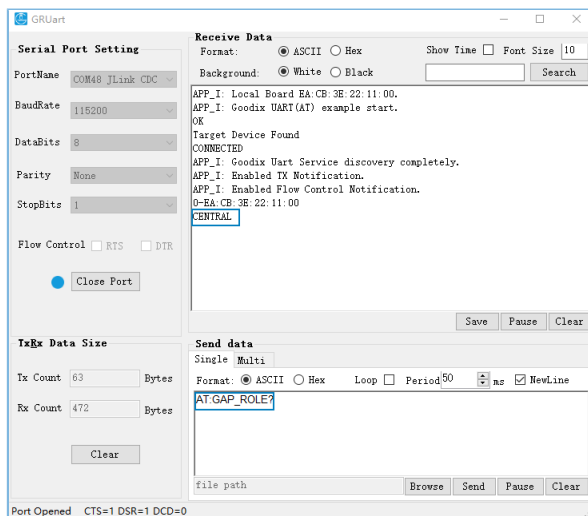


图 3-8 获取开发板A角色信息

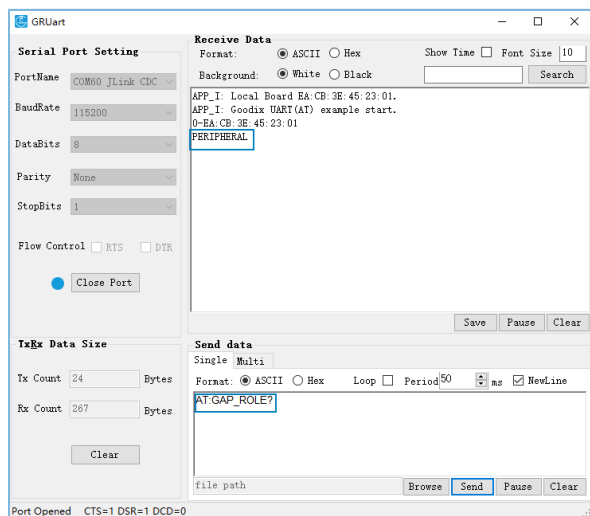


图 3-9 获取开发板B角色信息

#### 4. 利用透传服务进行数据传输。

- 开发板B（Server端）发送“Goodix\_BLE”消息到开发板A（Client端）

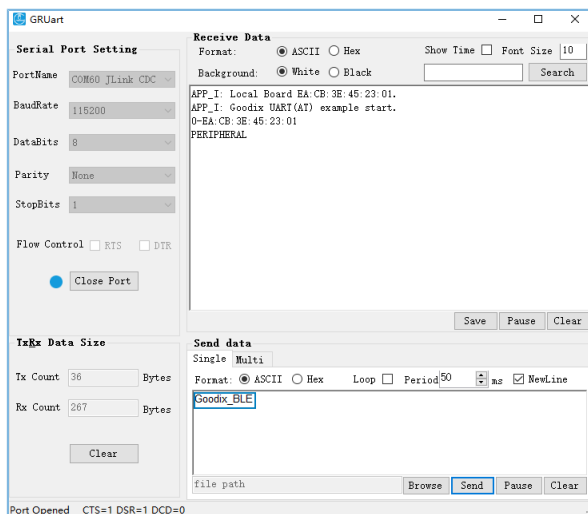


图 3-10 Server端发送数据

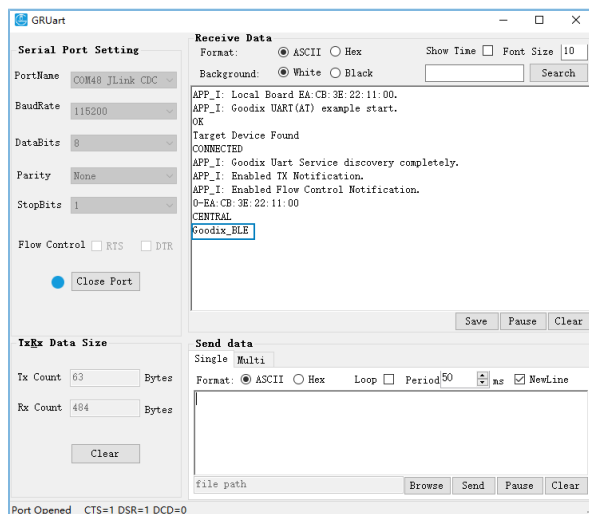


图 3-11 Client端接收数据

- 开发板A（Client端）发送“Hello Word!”消息到开发板B（Server端）

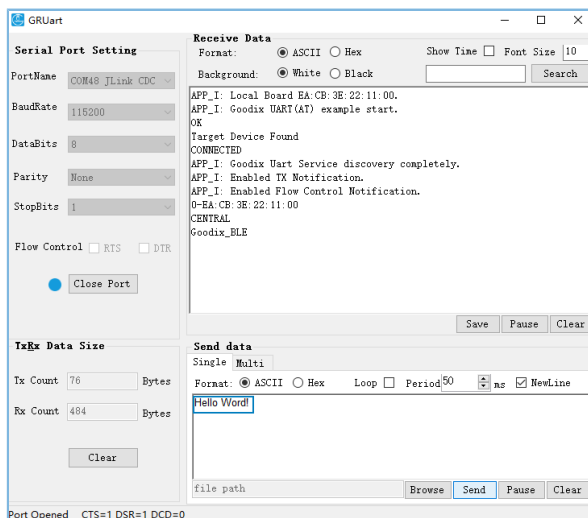


图 3-12 Client端发送数据

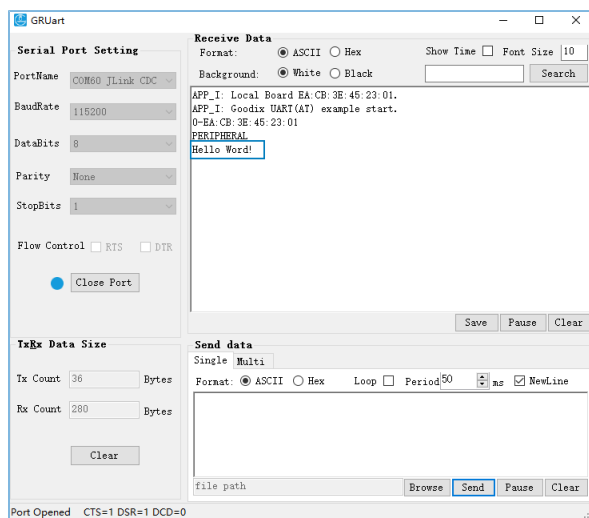


图 3-13 Server端接收数据

## 4 应用详解

本章将介绍ble\_app\_uart\_at示例的运行流程和关键代码。

### 4.1 运行流程

本节主要介绍ble\_app\_uart\_at示例的运行流程，以便用户深入了解其运行机制。

ble\_app\_uart\_at示例运行流程如下图所示：

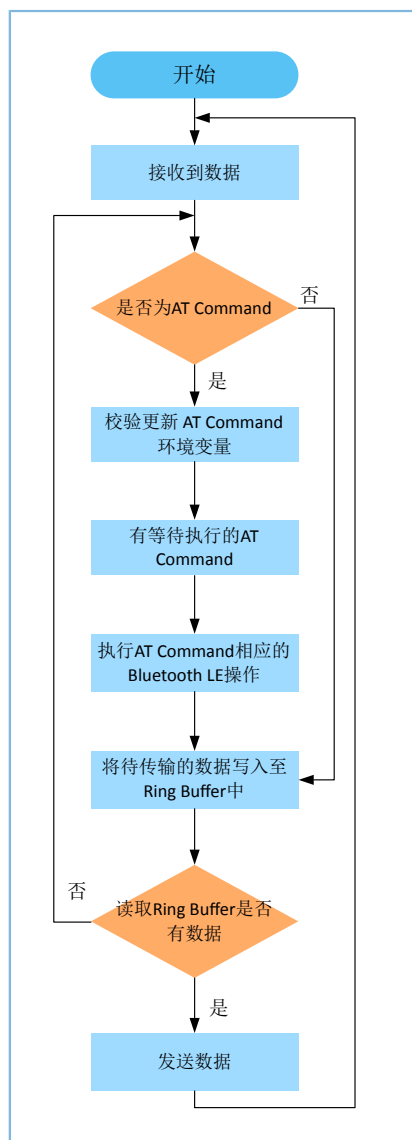


图 4-1 ble\_app\_uart\_at示例运行流程

1. 根据接收到的数据，判断是否为AT Command，若为AT Command则校验更新AT Command环境变量，否则执行步骤4。
2. 读取AT Command环境变量。若有待执行的AT Command且其Command Handler不为空，则执行步骤3。
3. 根据AT Command执行相应的Bluetooth LE操作。

4. 将待传输的数据写入Ring Buffer，其中Ring Buffer分为ble to uart buffer和uart to ble buffer两种，分别用来接收和发送数据。
  - 当利用串口工具发送AT Command时，会将AT Command执行结果缓存至ble to uart buffer中；
  - 当利用串口工具发送非AT Command数据时，以两块运行有ble\_app\_uart\_at固件的设备为例说明数据传输机制。两块开发板连接后，开发板A（Client端）利用串口工具向开发板B（Server端）发送数据（非AT Command）时，开发板A会将数据（非AT Command）缓存至uart to ble buffer中用于发送，开发板B会将数据缓存至ble to uart buffer中用于接收。
5. 读取Ring Buffer是否有数据，若有则读取并发送数据，否则继续循环。

AT Command执行流程如下图所示：

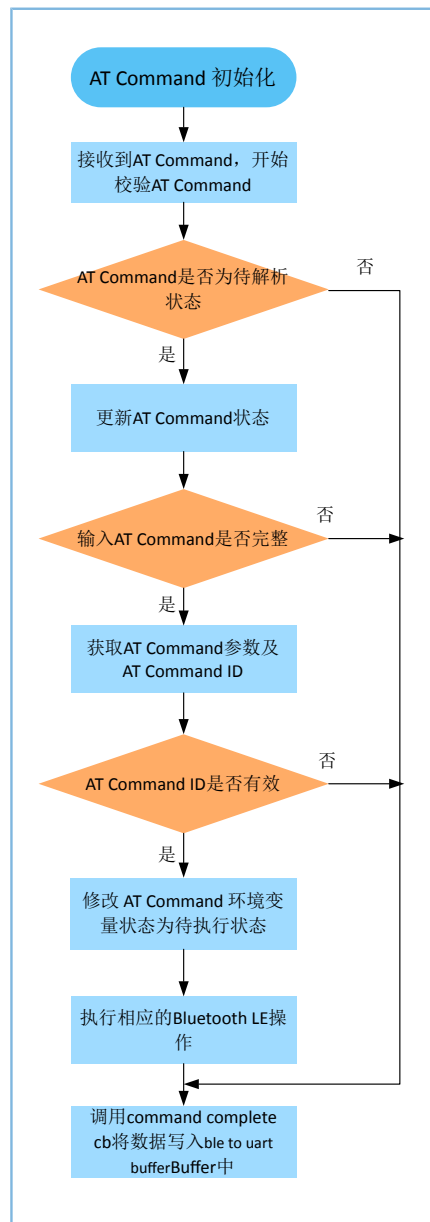


图 4-2 AT Command执行流程

1. AT Command初始化。完成AT Command属性表、command complete cb以及app timer的注册。

### 说明:

- AT Command属性表: 记录AT Command信息, 如AT Command ID、AT Command Tag、AT Command Tag Length、AT Command Handler。
  - command complete cb: 将AT Command执行结果写入至ble to uart buffer中。若command response有错误, 错误信息则为待传输数据, 否则command response data为待传输数据。
  - app timer: 超时管理。
2. 接收到AT Command后, 校验AT Command, 若为待解析状态, 则更新其状态为在解析状态并执行步骤3, 否则更新Error Code并执行步骤7。
  3. 校验输入的AT Command是否完整。完整的AT Command以AT:开始, 以\r\n结束。若校验成功则执行步骤4, 否则更新Error Code并执行步骤7。
  4. 获取AT Command的参数及ID。
  5. 校验AT Command ID是否有效, 若有效则更新AT Command状态为待执行状态, 否则更新Error Code并执行步骤7。
  6. 执行AT command。若AT Command Handler不为空则调用Handler执行相应的Bluetooth LE操作。否则更新Error Code并执行步骤7。
  7. 调用command complete cb, 将AT Command执行结果写入ble to uart buffer中。

## 4.2 关键代码

本章将详细介绍ble\_app\_uart\_at示例运行流程中的关键代码。

### 4.2.1 事件处理函数

路径: user\_app\user\_app.c

名称: gus\_service\_process\_event();

数据接收事件到来时, 判断接收到的数据是否为AT Command。若是AT Command, 则调用at\_cmd\_parse函数。否则将数据写入ble to uart buffer中。

```
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 校验更新AT Command环境变量

路径：工程目录下的gr\_libraries\at\_cmd.c

名称：at\_cmd\_parse();

如果接收到的数据为AT Command，则对AT Command环境变量进行校验和更新，若校验通过则将AT Command环境变量修改为待执行状态，具体代码如下：

```

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 执行AT Command对应的Bluetooth LE操作

路径：工程目录下的gr\_libraries\at\_cmd.c

名称：at\_cmd\_schedule();

读取AT Command环境变量，若有待执行的AT Command，且Command Handler不为空，则根据AT Command属性表执行相应的广播、扫描、连接等相关蓝牙操作。若AT Command为修改设备名，则调用uart\_at\_gap\_name\_set command handler修改设备名。

```

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);
    }
}
}

```

路径：工程目录下的user\_app\at\_cmd\_handler.c

名称：uart\_at\_gap\_name\_set();

修改设备名的代码如下所示：

```

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 将AT Command执行结果写入到ble to uart buffer中

路径：工程目录下的gr\_libraries\at\_cmd.c

名称：at\_cmd\_execute\_cplt();

AT Command Handler执行完成后，根据AT Command执行的返回值更新待传输的数据，并调用cmd\_cplt\_cb将AT Command执行结果写入ble to uart buffer，具体代码如下：

```
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;
    ...
}
```

路径：工程目录下的user\_app\at\_cmd\_handler.c

名称：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 读取Ring Buffer并发送数据

路径：工程目录下的user\_app\transport\_scheduler.c

名称：transport\_schedule();

在设备Notify、流控等已开启的前提下，若Ring Buffer中有数据，则先读取Ring Buffer中的数据，然后再将读取的数据进行发送。

```
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 自定义命令

本章描述了用户在使用及验证ble\_app\_uart\_at示例时，如何添加自定义的AT Command。

将自定义AT Command所需的AT Command ID、AT Command、AT Command Length以及AT Command Handler添加至AT Command属性表中，然后再实现Command Handler。

### 说明:

AT Command属性表所在路径为工程目录下user\_app\at\_cmd\_handler.c

例如，增加“交换MTU AT Command”的步骤如下：

1. 在`at_cmd.h`（位于SDK\_Folder\components\libraries\at\_cmd）的`at_cmd_id_t`结构体中添加所需的AT Command ID。
2. 在代码中更新AT Command属性表，将所需的AT Command信息添加至`s_at_cmd_attr_table`中。

AT Command属性表更新后如下所示：

```
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_cancle},
    {AT_CMD_DISCONN,         "DISCONN", 7,  uart_at_disconnect},
    {AT_CMD_MTU_EXCHANGE,    "MTU_EXC", 7,  uart_at_mtu_exchange},
};
```

### 说明:

加粗代码为新增代码。

3. 实现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 常见问题

### 6.1 设置GAP角色失败

- 问题描述  
使用AT Command设置GAP角色失败。
- 问题分析  
使用AT Command设置GAP角色时，若设备不处于STANDBY状态，可能会导致设置GAP角色失败。
- 处理方法  
利用AT Command设置GAP角色时，应确保设备处于STANDBY状态。

### 6.2 设置设备信息失败

- 问题描述  
使用AT Command设置设备信息失败。
- 问题分析  
使用AT Command设置设备信息时，如更改GAP角色、GAP名称等，其AT Command的“=”后可能有空格。
- 处理方法  
设置设备信息时，AT Command的“=”后不要有空格。

### 6.3 使用GRUart传输AT Command时，提示输入无效

- 问题描述  
使用GRUart传输AT Command时，提示输入无效。
- 问题分析  
AT Command以“\r\n”结尾，在使用GRUart传输AT Command时，可能未勾选“NewLine”。
- 处理方法  
在使用GRUart传输AT Command时，勾选“NewLine”。

7 附录

7.1 AT指令表

ble\_app\_uart\_at示例涉及的AT指令表如下所示：

表 7-1 AT指令表

AT指令类型	AT Command	描述	返回值	举例
AT测试	AT:TEST	测试AT指令功能是否正常。	返回OK	AT:TEST
版本	AT:VERSION?	获取版本号。	返回版本号	AT:VERSION?
重置系统	AT:RESET	重置系统。	-	AT:RESET
波特率	AT:BAUD= <NEW_VALUE>	设置波特率。  NEW_VALUE：波特率的值，取值范围为[0,2000000]。	成功返回OK  失败返回ERR: Invalid parameters.	AT:BAUD=4900
设备地址	AT:ADDR?	获取设备地址。	成功返回设备地址  失败无设备信息返回	AT:ADDR?
GAP角色	AT:GAP_ROLE?	获取设备的角色信息。	返回角色信息： NONE、OBSERVER、 BROADCASTER、 CENTRAL、 PERIPHERAL、 ALL	AT:GAP_ROLE?
	AT:GAP_ROLE= <NEW_ROLE>	设置设备的角色。  NEW_ROLE：设置的设备角色，可选值为N,n,O,o,B,b,C,c,P,p,A,a。	成功返回OK  失败返回ERR: Command request is not allowed.	AT:GAP_ROLE=O
GAP名称	AT:GAP_NAME?	获取设备名。	成功返回设备名  失败返回相应错误信息	AT:GAP_NAME?
	AT:GAP_NAME=<INDEX, NEW_NAME>	设置设备名。  INDEX：设备名的写权限，取值范围为[0,4]。 <ul style="list-style-type: none"><li>0表示禁止写</li><li>1表示链路不加密不认证</li><li>2表示链路加密但不认证</li></ul>	成功返回OK  失败返回相应错误信息	AT:GAP_NAME=1,Goodix



AT指令类型	AT Command	描述	返回值	举例
		<ul style="list-style-type: none"> <li>3表示链路加密且认证（MITM）</li> <li>4表示链路加密且认证（secure connectisns）</li> </ul> NEW_NAME: 自定义广播名。		
广播	AT:ADV_PARAM=<ADV_INTERVAL, ADV_DURATION>	设置广播参数。 ADV_INTERVAL: 广播间隔, 以0.625 ms为单位, 取值范围为> 32。 ADV_DURATION: 广播时长, 以10 ms为单位, 值为0时表示持续广播直至host关闭它。广播类型为有限可发现模式时, 取值范围为[1,18000]; 定向高占空比模式时, 取值范围为[1,128]。	成功返回OK 失败返回相应错误信息	AT:ADV_PARAM=80,0
	AT:ADV_START	开启广播。	成功返回OK 失败返回相应错误信息	AT:ADV_START
	AT:ADV_STOP	停止广播。	成功返回OK 失败返回相应错误信息	AT:ADV_STOP
扫描	AT:SCAN_PARAM=<SCAN_INTERVAL, SCAN_DURATION>	设置扫描参数。 SCAN_INTERVAL: 扫描间隔, 以0.625 ms为单位, 取值范围为[4,16384]。 SCAN_DURATION: 扫描时长, 以0.625 ms为单位, 取值范围为[1,65535]	成功返回OK 失败返回相应错误信息	AT:SCAN_PARAM=176,1000
	AT:SCAN_START	开启扫描。	成功返回OK 失败返回相应错误信息	AT:SCAN_START
	AT:SCAN_STOP	停止扫描。	成功返回OK 失败返回相应错误信息	AT:SCAN_STOP
连接	AT:CONN_PARAM=	设置连接参数。	成功返回OK	AT:CONN_PARAM=

AT指令类型	AT Command	描述	返回值	举例
	<CONN_INTERVAL, CONN_LATENCY, CONN_SUP_TIMEOUT>	CONN_INTERVAL: 连接间隔, 取值范围为[6,3200]。 CONN_LATENCY: 可以忽略的连接事件数, 取值范围< (CONN_SUP_TIMEOUT/CONN_INTERVAL) -1 CONN_SUP_TIMEOUT: 监控超时, 取值范围为[10,3200]。	失败返回相应错误信息	12,5,3200
	AT:CONN_INIT=	发起连接。	成功返回CONNECTED 失败返回相应错误信息	AT:CONN_INIT=
	AT:CONN_CANCEL	停止连接。	成功返回OK 失败返回相应错误信息	AT:CONN_CANCEL
断连	AT:DISCONN	断开连接。	成功返回DISCONNECTED 失败返回相应错误信息	AT:DISCONN
交换MTU	AT:MTU_EXC	交换MTU。	成功返回MTU值 失败返回相应错误信息	AT:MTU_EXC

## 7.2 Error Code详解

使用AT Command时失败返回的错误信息（Error Code）如下表所示。

表 7-2 Error Code详解

名称	描述
AT_CMD_ERR_INVALID_INPUT	输入信息无效
AT_CMD_ERR_UNSUPPORTED_CMD	不支持所输入的AT指令
AT_CMD_ERR_PARSE_NOT_ALLOWED	AT Command处于不可解析状态
AT_CMD_ERR_CMD_REQ_ALLOWED	命令请求不被允许, 设置GAP角色时, 若设备的状态不为STANDBY, 则返回该错误

名称	描述
AT_CMD_ERR_NO_CMD_HANDLER	AT Command Handler为空时返回该错误
AT_CMD_ERR_INVALID_PARAM	输入的AT指令参数无效
AT_CMD_ERR_HAL_ERROR	Hal层操作超时返回该错误
AT_CMD_ERR_TIMEOUT	AT指令执行超时
AT_CMD_ERR_OTHER_ERROR	其他错误