# GR551x DFU Application Note

**Version: 1.7**

**Release Date: 2020-12-25**

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

**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

# Preface

**Purpose**

This document introduces principles and applications for Device Firmware Update (DFU) of GR551x SoCs, to help you quickly get familiar with GR551x firmware update modes, and get started with secondary development.

**Audience**

This document is intended for:

- GR551x user

- GR551x developer

- GR551x tester

- Technical writer

**Release Notes**

This document is the fifth release of *GR551x DFU Application Note*, corresponding to GR551x SoC series.

**Revision History**

| Version | Date | Description |
|---------|------|-------------|
| 1.0 | 2019-12-08 | Initial release |
| 1.3 | 2020-03-16 | Updated the release time in the footers. |
| 1.5 | 2020-05-30 | Updated the structure figure of Boot_Info sector in flash in "DFU Storage"; optimized the descriptions of "DFU Mode". |
| 1.6 | 2020-06-30 | Updated the data contents sent from the Host in "Configure External Flash Command". |
| 1.7 | 2020-12-25 | • Updated the contents in "Steps" and "Why does DFU for Application Firmware on a Bare Device Fail?";<br>• Added "Why do Exceptions Occur in OTA DFU for Application Firmware on an OS Device?". |

# Contents

# 1 Introduction

Device Firmware Update (DFU) is a boot loading mechanism to update firmware for target devices, allowing developers to quickly fix defects and provide more functions for their products.

This document introduces the methods and principles of DFU for GR551x SoCs, and how to enable and test DFU functionalities.

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

Table 1-1 Reference documents

| Name | Description |
| --- | --- |
| GR551x Developer Guide | Introduces the software/hardware and quick start guide of GR551x SoCs. |
| Bluetooth Core Spec v5.1 | Offers official Bluetooth standards and core specification (v5.1) from Bluetooth SIG. Available at https://www.bluetooth.com/specifications/bluetooth-core-specification/. |
| J-Link/J-Trace User Guide | Provides J-Link operational instructions. Available at www.segger.com/downloads/jlink/ UM08001_JLink.pdf. |
| Keil User Guide | Offers detailed Keil operational instructions. Available at www.keil.com/support/man/docs/uv4/ . |
| GR551x OTA Example Application | Introduces how to implement Over The Air (OTA) for GR551x firmware on GRToolbox (Android). |
| GR551x Bluetooth Low Energy Stack User Guide | Introduces the BLE Protocol Stack supported by GR551x SoCs. |

# 2 GR551x DFU

This chapter introduces fundamental concepts about GR551x DFU from the following aspects:

- DFU Mode

- DFU Device Role

- DFU File Format

- DFU Communications Protocol

- DFU Command Set

## 2.1 DFU Mode

GR551x SoCs support two DFU approaches (update through Bluetooth LE connection):

- Approach 1: Perform update in application firmware. Run the application firmware to download the target firmware. After download, the system jumps to the target firmware for running. During update, users can update firmware directly without disabling Bluetooth connection on a mobile phone. This approach provides smooth user experience.

- Approach 2: Jump to Boot firmware for firmware update. During firmware update, the system jumps from applications to the Boot firmware, and the Boot firmware enables download of the target firmware. After download, the system jumps to the target firmware for running. During update, it is required to first disable Bluetooth connection on the mobile phone and then scan and reconnect to the Boot firmware via Bluetooth. Compared with the first approach, this approach makes the best use of flash.

---

📖 **Note**:

- To choose the Approach 1, you need to pre-define the load addresses of the current application firmware and the target firmware, and set different code load addresses for the current application firmware and the target firmware.

- The Approach 1 supports "Copy Mode". That is, before DFU, first update the target firmware to another unused flash address; then copy it to the required run address. For details, see "Connect to GR5515 SK Board and Update Firmware" in *GR551x OTA Example Application*.

---

## 2.2 DFU Device Role

Two DFU device roles are defined:

- Control device (the host): a device, such as a mobile phone, that sends update data to the target device

- Target device (the device): a device, such as a wristband, that receives update data from the control device
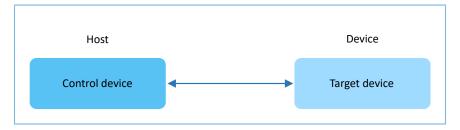
Figure 2-1 DFU device role

## 2.3 DFU File Format

Firmware files transferred in DFU mode are in BIN format, including encrypted and non-encrypted BIN files.
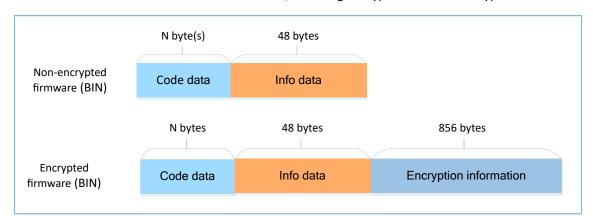


Figure 2-2 Data format of BIN files

Each field of the data format is detailed as below:

- Code data: firmware data that shall be 16-byte aligned. N indicates a variable length.

- Info data: firmware descriptions

- Encryption information: information used in encrypting non-encrypted firmware

The info data format (little-endian mode) is shown in the table below:

Table 2-1 Info data format

| Byte | Field | Description | | |
|------|-------|-------------|---|---|
| 0–1 | pattern | Data identifier for SoCs, value: 0x4744 | | |
| 2–3 | version | Version information | | |
| 4–7 | bin_size | Code data length (byte) | | Boot info (24 bytes) |
| 8–11 | check_sum | Checksum of code data bit | | |
| 12–15 | load_addr | Start address of code data storage area | | |
| 16–19 | run_addr | Run address of code data | | |
| 20–23 | xqspi_xip_cmd | SPI access mode | | |
| 24–27 | boot config | Bit field | [0:3]: clock speed | |

| Byte | Field | Description | |
|------|-------|-------------|---|
| | | [4]: code copy mode | |
| | | [5:7]: system clock | |
| | | [8]: check image | |
| | | [9]: boot delay time | |
| | | [10:31]: reserved | |
| 28–39 | comments | Firmware descriptions | |
| 40–47 | reserved | Used for 16-byte alignment; value: 0x00 | |

## 2.3.1 DFU Storage

The firmware information is stored in the Img_Info area in GR551x flash. This area (start address: 0x01000040; length: 400 bytes) can store up to 10 firmware packages.
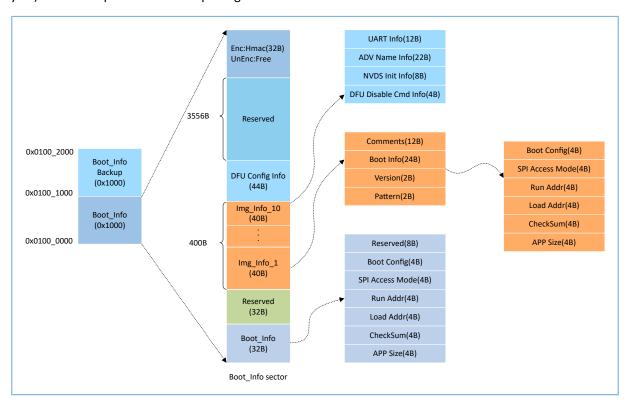


Figure 2-3 Structure of Boot_Info sector in flash

# 2.4 DFU Communications Protocol

The firmware update between the host and the device is based on DFU communications protocols.

## 2.4.1 Basic Frame

The basic frame defines the lowest-level data packet structure in communications. The application data packet protocol is based on the basic frame, in the "Data" field of the basic frame. If the basic frame length exceeds the

maximum payload of link communications, the host needs to segment the frame for transmission. After receiving the correct frame header and data length, the device starts processing data.

## 2.4.2 Frame Structure

| 2 | 2 | 2 | 0–2048 | 2 | bytes |
|---|---|---|---|---|---|
| Frame header | Frame type | Data length | Data | Parity bit | |
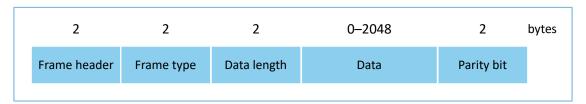
Figure 2-4 Frame structure

- Frame header: the start of a frame, represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D'

- Frame type: used to distinguish data types in the "Data" field

- Data length: length of data in the "Data" field

- Data: data with configurable length; maximum length: 2048 bytes

- Parity bit: 16-bit checksum for frame type, response, data length, and data

## 2.4.3 Byte Order

The little-endian mode is adopted for the "Data" field of the basic frame. The low byte data shall be stored at low addresses in flash, and the high byte data at high addresses.

## 2.5 DFU Command Set

DFU commands are delivered by the host and received by the device.

Table 2-2 DFU command description

| Command | Command Code | Description |
|---|---|---|
| Program Start | 0x0023 | The host sends Image Info when programs are downloaded to the device. |
| Program Flash | 0x0024 | Write up to 1024 bytes of data for one time to a specified address of flash. This command can specify the way to write data to flash (erasable write/non-erasable write). |
| Program End | 0x0025 | The host sends this command to notify the device that the programming data has been sent. The first byte of the data field is Reset Flag, which indicates the device to reset and run the current application or not after the programming data is sent. |
| Configure External Flash | 0x002A | Configure external flash. |
| Get Flash Information | 0x002B | Get flash information. |

## 2.5.1 Program Start Command

The host sends this command to send the Image Info (data excluding the "reserved" field; 40 bytes) to the device.

After the device receives the Image Info:

- For internal flash programming, check whether the Code Load Address of the programming data is the internal flash address.

- For external flash programming, specify the programming address and the data length.

### 2.5.1.1 Data Sent from the Host

Table 2-3 Format of data sent through the Program Start command

| Byte No. | Description | | Valid Value | Note |
|---|---|---|---|---|
| 0–1 | Frame header | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | 0x0023 | Program Start command |
| 4–5 | Data length | | 0x0029/0x0009 | • If the Program Start command targets firmware, the "Data" field has 41 bytes, including 1 byte for flash type and 40 bytes for Image Info.<br>• If the Program Start command targets data, the "Data" field has 9 bytes, including 1 byte for flash type, 4 bytes for start address, and 4 bytes for data content. |
| 6 | Data | Flash type | 0x00/0x01 | • 0x00: internal flash<br>• 0x01: external flash |
| 7–14 or 7–46 | | Data to be written to flash | Value range for each byte: 0x00–0xFF | Data content |
| 15–16 or 47–48 | Checksum | | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

### 2.5.1.2 Response Data from the Device

Table 2-4 Format of data replied through the Program Start command

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 0–1 | Frame header | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | 0x0023 | Program Start command |
| 4–5 | Data length | 0x0001 | Length of data in the "Data" field |
| 6 | Response | 0x01/0x02 | • 0x01: Data send succeeds.<br>• 0x02: Data send fails. |

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 7–8 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and response field |

## 2.5.2 Program Flash Command

The host sends this command to write data to a valid (internal or external) flash address of the device. After receiving the command, the device parses the start address, length, and content of the data. If the start address is valid, the device starts writing data from this address (see Table 2-5 for data format), and returns ACK. Otherwise, the device returns NACK.

### 2.5.2.1 Data Sent from the Host

Table 2-5 Format of data sent through the Program Flash command

| Byte No. | Description | | Valid Value | Note |
|---|---|---|---|---|
| 0–1 | Frame header | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | 0x0024 | Program Flash command |
| 4–5 | Data length | | 0x0007–0x0800 | Length of data in the "Data" field |
| 6 | Data | Program type | 0x00/0x01/0x02/ 0x10/0x11/0x12 | <ul><li>0x00: Store data after erasing the internal flash page at a specified address.</li><li>0x01: Store data in internal flash according to Image Info sent by the Program Start command.</li><li>0x02: Call a flash write API to write data to internal flash.</li><li>0x10: Store data after erasing the external flash page at a specified address.</li><li>0x11: Store data in external flash according to Image Info sent by the Program Start command.</li><li>0x12: Call a flash write API to write data to external flash.</li></ul> |
| 7–10 | | Start address | Value range for each byte: 0x00–0xFF | Valid flash address of the device |
| 11–12 | | Length of data to be written to flash | 0x0000–0x00FF | Maximum length of data to be written to flash: 1024 bytes |
| 13–N | | Data to be written to flash | Value range for each byte: 0x00–0xFF | Data to be written to flash |
| N+1 to N+2 | Checksum | | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

N in Table 2-5 indicates variable length for the "Data" field, ranging from 14 to 1036.

### 2.5.2.2 Response Data from the Device

Table 2-6 Format of data replied through the Program Flash command

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 0–1 | Frame header | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | 0x0024 | Program Flash command |
| 4–5 | Data length | 0x0001 | 1 byte (for response) |
| 6 | Response | 0x01/0x02 | • 0x01: Data write succeeds.<br>• 0x02: Data write fails. |
| 7–8 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and response field |

## 2.5.3 Program End Command

The host sends this command to notify the device that the programming data has been sent. The data field contains the start address after the next reset and the Reset Flag. The Reset Flag decides whether the device runs the downloaded application or not immediately after receiving the Program End command.

### 2.5.3.1 Data Sent from the Host

Table 2-7 Format of data sent through the Program End command

| Byte No. | Description | | Valid Value | Note |
|---|---|---|---|---|
| 0–1 | Frame header | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | 0x0025 | Program End command |
| 4–5 | Data length | | 0x0005 | Length of data in the "Data" field |
| 6 | Data | Reset Flag | 0x00/0x01/0x02/0x12 | • 0x00: Store the firmware information in the img_info area in flash.<br>• 0x01: Store the firmware information in the img_info and boot_info areas in flash, and run the programming firmware after reset.<br>• 0x02: Download data to internal flash without operations to the img_info and boot_info areas.<br>• 0x12: Download data to external flash. |

| Byte No. | Description | | Valid Value | Note |
|---|---|---|---|---|
| 7–10 | | Checksum of programming file | Value range for each byte: 0x00–0xFF | Checksum of BIN file |
| 11–12 | Checksum | | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

### 2.5.3.2 Response Data from the Device

Table 2-8 Format of data replied through the Program End command

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 0–1 | Frame header | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | 0x0025 | Program End command |
| 4–5 | Data length | 0x0001 | 1 byte (for response) |
| 6 | Response | 0x01/0x02 | • 0x01: Notification succeeds.<br>• 0x02: Notification fails. |
| 7–8 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and response field |

## 2.5.4 Read/Update Data in System Configuration Area Command

The host sends this command to handle data in the System Configuration Area of the device, including reading and updating the data. For addresses of System Configuration Area, see "Section 2.3.1 DFU Storage".

### 2.5.4.1 Data Sent from the Host

Table 2-9 Format of data sent through the Read/Update Data in System Configuration Area command

| Byte No. | Description | | Valid Value | Note |
|---|---|---|---|---|
| 0–1 | Frame header | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | 0x0027 | Read/Update Data in System Configuration Area command |
| 4–5 | Data length | | 0x0007–0x0407 | Length of data in the "Data" field |
| 6 | Data | Operating command | 0x00/0x01 | • 0x00: Read the data in the System Configuration Area.<br>• 0x01: Update the data in the System Configuration Area. |
| 7–10 | | Start address | 0x01000000–0x01002000 | A valid address in the System Configuration Area |
| 11–12 | | Data length | 0x0000–0x0400 | Length of data to be read or updated |
| 13–N | | Update data | Value range for each byte: 0x00–0xFF | N/A for data read command |

| Byte No. | Description | Valid Value | Note |
|----------|-------------|-------------|------|
| N+1 to N+2 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

📖 **Note**:

N in Table 2-9 indicates variable length for the "Data" field:

- For a data update command, N ranges from 14 to 1036.

- For a data read command, the "Data" field has a fixed length of 7 bytes.

## 2.5.4.2 Response Data from the Device

Table 2-10 Format of data replied through the Read/Update Data in System Configuration Area command

| Byte No. | Description | | Valid Value | Note |
|----------|-------------|---|-------------|------|
| 0–1 | Frame header | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | 0x0027 | Read/Update Data in System Configuration Area command |
| 4–5 | Data length | | 0x0002–0x0402 | Length of data in the "Data" field |
| 6 | Data | Response | 0x01/0x02 | • 0x01: Data handling succeeds.<br>• 0x02: Data handling fails. |
| 7 | | Operating command | 0x00/0x10<br>0x01/0x11 | • 0x00: Read the data in the System Configuration Area (non-encrypted SoCs).<br>• 0x10: Read the data in the System Configuration Area (encrypted SoCs).<br>• 0x01: Update the data in the System Configuration Area (non-encrypted SoCs).<br>• 0x11: Update the data in the System Configuration Area (encrypted SoCs). |
| 8–11 | | Start address of the System Configuration Area | 0x01000000–0x01002000 | N/A for data update command |
| 12–13 | | Length of data in the System Configuration Area | 0x0000–0x0400 | |
| 14–N | | Content of data in the System Configuration Area | Value range for each byte: 0x00–0xFF | |

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| N+1 to N+2 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

## 2.5.5 Configure External Flash Command

The host sends this command to configure the external flash SPI of the device.

### 2.5.5.1 Data Sent from the Host

Table 2-11 Format of data sent through the Configure External Flash command

| Byte No. | Description | | Valid Value | Note |
|---|---|---|---|---|
| 0–1 | Frame header | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | 0x002A | Configure External Flash command |
| 4–5 | Data length | | 0x0014 | Length of data in the "Data" field |
| 6 | | External flash type | 0x01/0x02 | • 0x01: SPI flash<br>• 0x02: QSPI flash |
| 7–9 | | CS IO TYPE | 00–03 | GPIO type selection |
| | | CS PIN | 00–31 | GPIO pin selection |
| | | CS IO MUX | 00–09 | Pin Mux selection |
| 10–12 | | CLK IO TYPE | 00–03 | GPIO type selection |
| | | CLK PIN | 00–31 | GPIO pin selection |
| | | CLK IO MUX | 00–09 | Pin Mux selection |
| 13–15 | | MOSI(IO0) IO TYPE | 00–03 | GPIO type selection |
| | | MOSI(IO0) PIN | 00–31 | GPIO pin selection |
| | Data | MOSI(IO0) IO MUX | 00–09 | Pin Mux selection |
| 16–18 | | MOSI(IO1) IO TYPE | 00–03 | GPIO type selection |
| | | MOSI(IO1) PIN | 00–31 | GPIO pin selection |
| | | MOSI(IO1) IO MUX | 00–09 | Pin Mux selection |
| 19–21 | | IO2 IO TYPE | 00–03 | GPIO type selection, valid for QSPI only |
| | | IO2 PIN | 00–31 | GPIO pin selection, valid for QSPI only |
| | | IO2 IO MUX | 00–09 | Pin Mux selection, valid for QSPI only |
| 22–24 | | IO3 IO TYPE | 00–03 | GPIO type selection, valid for QSPI only |
| | | IO3 PIN | 00–31 | GPIO pin selection, valid for QSPI only |
| | | IO3 IO MUX | 00–09 | Pin Mux selection, valid for QSPI only |
| 25 | | QSPI ID | 00–02 | QSPI Module ID, valid for QSPI only |

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 26–27 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

### 2.5.5.2 Response Data from the Device

Table 2-12 Format of data replied through the Configure External Flash command

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 0–1 | Frame header | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | 0x002A | Initialize external flash commands. |
| 4–5 | Data length | 0x0001 | 1 byte (for response) |
| 6 | Response | 0x01/0x02 | • 0x01: Configuration succeeds.<br>• 0x02: Configuration fails. |
| 7–8 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and response field |

## 2.5.6 Get Flash Information Command

The host sends this command to get internal/external flash information from the device, including flash ID and flash size. External flash size is available through the Serial Flash Discoverable Parameters (SFDP) protocol. For all flash chips supporting the SFDP protocol, the host can get the flash size by sending this command.

### 2.5.6.1 Data Sent from the Host

Table 2-13 Format of data sent through the Get Flash Information command

| Byte No. | Description | Valid Value | Note |
|---|---|---|---|
| 0–1 | Frame header | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | 0x002B | Get flash ID. |
| 4–5 | Data length | 0x0001 | 1 byte |
| 6 | Flash type | 0x00/0x01 | • 0x00: internal flash<br>• 0x01: external flash |
| 7–8 | Checksum | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

### 2.5.6.2 Response Data from the Device

Table 2-14 Format of data replied through the Get Flash Information command

| Byte No. | Description | | | Valid Value | Note |
|---|---|---|---|---|---|
| 0–1 | Frame header | | | 0x4744 | Represented by 0x47 and 0x44 which are the ASCII code values of characters 'G' and 'D' |
| 2–3 | Frame type | | | 0x002B | Get flash ID. |
| 4–5 | Data length | | | 0x0009 | 9 bytes (for response) |
| 6 | Data | Response | | 0x01/0x02 | • 0x01: Data get succeeds.<br>• 0x02: Data get fails. |
| 7–14 | | Flash information | Flash ID | Value range for each byte: 0x00–0xFF | Flash ID (4 bytes) |
| | | | Flash size | Value range for each byte: 0x00–0xFF | Flash size (4 bytes) |
| 15–16 | Checksum | | | 0x0000–0xFFFF | 16-bit checksum for frame type, data length, and data field |

# 3 Enablement of DFU

This chapter introduces how to enable DFU for GR551x SoCs.

## 3.1 Add DFU to Application Firmware

The DFU functionality is encapsulated in the `SDK_Folder\components\libraries\dfu_port` module. To add DFU to applications, you only need to call related APIs. Details are provided below by taking ble_dfu_boot as an example.

---

📖 **Note**:

SDK_Folder is the root directory of GR551x SDK.

---

### 3.1.1 ble_dfu_boot Project

The source code and project file of the example ble_dfu_boot are in

`SDK_Folder\projects\ble\dfu\ble_dfu_boot`, and project file is in the Keil_5 folder.

Double-click the project file, *ble_dfu_boot.uvprojx*, to view the project directory structure of ble_dfu_boot in Keil. For related files, see Table 3-1.

Table 3-1 File description of ble_dfu_boot

| Group | File | Description |
|---|---|---|
| gr_profiles | ble_prf_utils.c | This file contains profile-related operational tools. |
| | ota.s | This file implements OTA Service. |
| user_callback | user_gap_callback.c | This file implements GAP callback, such as connection, disconnection, and GAP parameter update. |
| | user_gatt_common_callback.c | This file implements GATT common callback, such as MTU exchange. |
| user_platform | user_periph_setup.c | This file configures the serial port parameters, device address, power management mode, and DFU. |
| user_app | main.c | This file contains the main() function. |
| | user_app.c | This file implements OTA profile registration and logical processing. |

### 3.1.2 Steps

1. Configure the interfaces for DFU. Two approaches are available: Update through Bluetooth LE connection and update through UART. Users can choose as needed.

   - Update through Bluetooth LE connection.

     **Path:** `user\user_app.c` under the project directory

     **Name:** services_init();

     Call dfu_service_init() in this function to initialize the Bluetooth LE services for update.

```
static void services_init(void)
{
    dfu_service_init(NULL);
}
```

**Path:** `user_callback\user_gatt_common_callback.c` under the project directory

**Name:** app_gatt_mtu_exchange_cb();

Call dfu_ble_set_mtu_size() in this function to configure the MTU value for update through Bluetooth LE connection.

```
static void app_gatt_mtu_exchange_cb(uint8_t conn_idx, uint8_t status, uint16_t mtu)
{
    if(BLE_SUCCESS == status)
    {
        dfu_ble_set_mtu_size(mtu);
    }
}
```

**Path:** `platform\user_periph_setup.c` under the project directory

**Name:** app_periph_init();

Call dfu_port_init() in this function to initialize the DFU module.

```
void app_periph_init(void)
{
SYS_SET_BD_ADDR(s_bd_addr);
bsp_uart_init();
app_log_assert_init();
pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
dfu_port_init(NULL,NULL);
}
```

- Update through UART.

    **Path:** `platform\user_periph_setup.c` under the project directory

    **Name:** app_uart_evt_handler();

    Call app_uart_receive_async() and dfu_uart_receive_data_process() in the APP_UART_EVT_RX_DATA event to update through UART.

```
void app_uart_evt_handler(app_uart_evt_t * p_evt)
{
    switch(p_evt->type)
    {
        case APP_UART_EVT_TX_CPLT:
            break;
        case APP_UART_EVT_RX_DATA:
            app_uart_receive_async(APP_UART_ID_0, uart_rx_data, UART_RX_SIZE);
            dfu_uart_receive_data_process(uart_rx_data, p_evt->data.size);
            break;
        case APP_UART_EVT_ERROR:
            break;
        default:break;
```

```
    }
```

**Path:** `platform\user_periph_setup.c` under the project directory

**Name:** app_periph_init();

Initialize UART and the DFU module in this function (it is required to mount uart_send_data in the DFU module), and call the UART asynchronous receiving API.

```
void app_periph_init(void)
{
    SET_BD_ADDR(BD_ADDR_NVDS_TAG, BD_ADDR_LENGTH, s_bd_addr);
    bsp_uart_init();
    app_uart_receive_async(APP_UART_ID_0, uart_rx_data, UART_RX_SIZE);
    dfu_port_init(uart_send_data, &dfu_pro_call);
    pwr_mgmt_mode_set(PMR_MGMT_ACTIVE_MODE);
}
```

Table 3-2 Input parameter description for dfu_port_init()

| Parameter | Description | Value |
|-----------|-------------|-------|
| uart_send_data | Enable/Disable DFU through UART | Yes: The UART send function which is implemented by users is assigned to this parameter. No: **NULL** is assigned to this parameter. |
| dfu_pro_call | Enable/Disable monitoring on update status at the application layer | Yes: The DFU status handling callback function is assigned to this parameter. No: **NULL** is assigned to this parameter. |

2. Call dfu_schedule() in the while(1) { } loop of the main() function.

**Path:** `user\main.c` under the project directory

**Name:** main();

Call dfu_schedule() in this function to schedule tasks.

```
int main (void)
{
    // Initialize user peripherals.
    app_periph_init();

    // Initialize BLE Stack.
    ble_stack_init(&s_app_ble_callback, &heaps_table);

    // loop
    while (1)
    {
        dfu_schedule();
#if SK_GUI_ENABLE
        gui_refresh_schedule();
#endif
        pwr_mgmt_schedule();
    }
}
```

---

📖 **Note**:

To implement DFU on an OS device, it is recommended to create a task for DFU to implement dfu_schedule, and the task stack assigned shall be no less than 6 KB, otherwise task stack overflow may occur.

---

## 3.2 Jump to Boot Firmware for Firmware Update

To jump to Boot for firmware update, you need to download the DFU Boot firmware (such as ble_dfu_boot) to be updated to a GR551x SoC. Update applications then jump to DFU Boot for firmware update. Details are provided below by taking ble_app_template_dfu as an example.

The source code and project file of the project ble_app_template_dfu are in `SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu`, and project file is in the Keil_5 folder.

Double-click the project file, *ble_app_template_dfu.uvprojx* in Keil for compilation. Information about the DFU Boot firmware displays in the log pane of Keil.

**Path:** `user\user_app.c` under the project directory

**Name:** dfu_enter();

When receiving an update command, applications call dfu_start_address() in this function to load boot information of the DFU Boot firmware.

```
static void dfu_enter(void)
{
    //use flash dfu boot
    boot_info_t boot_info =
    {
    .bin_size = 0x26cf0,
    .check_sum = 0xf04eff,
    .load_addr = 0x1002000,
    .run_addr = 0x1002000,
    .xqspi_xip_cmd = 0xeb,
    .xqspi_speed = 0x0,
    .code_copy_mode = 0x0,
    .system_clk = 0x0,
    .check_image = 0x0,
    .boot_delay = 0x1,
    .is_dap_boot = 0x1,
     };
    dfu_start_address(&boot_info);
}
```

# 4 GR551x OTA DFU

The host can implement OTA firmware update for target devices through wireless transmission. To update through Bluetooth LE connection, set the OTA Profile and the OTA Service.

## 4.1 Bluetooth LE OTA Profile

The Goodix-customized OTA Profile defines two device roles:

- Host: the Central that sends user application and fixed user data, including font library/image data and external application data. The Central can be a PC, a mobile phone, or other devices.

- Device: the Peripheral that receives user application and fixed user data. The Peripheral can be a wristband, a heart rate meter, or other devices.
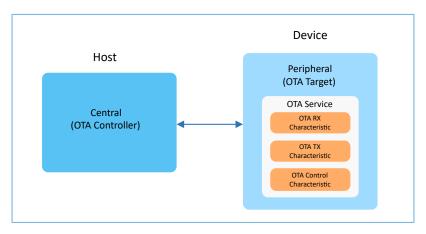


Figure 4-1 OTA Profile device roles

## 4.2 Bluetooth LE OTA Service

The Goodix-customized OTA Service provides necessary information for DFU. The OTA Service is independent and supports the following GATT operations:

- Write Characteristic Value

- Notifications

- Read Characteristic Descriptors

- Write Characteristic Descriptors

### 4.2.1 OTA Service and OTA Characteristics

The UUID of OTA Service is a6ed0401-d344-460a-8075-b9e8ec90d71b. OTA characteristics are divided into 3 types, as listed in Table 4-1.

Table 4-1 OTA characteristics

| Description | UUID | Property |
|---|---|---|
| OTA RX Characteristic | a6ed0402-d344-460a-8075-b9e8ec90d71b | Write without Response |

| Description | UUID | Property |
|---|---|---|
| OTA TX Characteristic | a6ed0403-d344-460a-8075-b9e8ec90d71b | Notify, Indicate |
| OTA Control Characteristic | a6ed0404-d344-460a-8075-b9e8ec90d71b | Write without Response, Indicate |

The role of each characteristic:

- OTA RX Characteristic: Receive data from the host with a property of Write without Response to speed up data transmission.

- OTA TX Characteristic: Send data to the host with a property of Notify. Data is reported by the device to the host in this way.

- OTA Control Characteristic: Receive control commands from the host, such as a command to start DFU process for a device.

## 4.3 OTA DFU Test

To perform an OTA DFU test, see *GR551x OTA Example Application* for details.

# 5 DFU Test Through UART with GR551x

This chapter introduces how to perform DFU tests through UART by using a GR5515 Starter Kit Board (GR5515 SK Board).

## 5.1 Preparation

Perform the following tasks before the test.

- **Hardware preparation**

Table 5-1 Hardware preparation

| Name | Description |
|---|---|
| J-Link debug probe | JTAG emulator launched by SEGGER. For more information, visit www.segger.com/products/debug-probes/j-link/. |
| Development board | GR5515 Starter Kit Board (2 boards in total) |

- **Software preparation**

Table 5-2 Software preparation

| Name | Description |
|---|---|
| Windows | Windows 7/Windows 10 |
| J-Link driver | A J-Link driver. Available at www.segger.com/downloads/jlink/. |
| Keil MDK5 | An integrated development environment (IDE). Available at www.keil.com/download/product/. |
| GProgrammer (Windows) | A GR551x programming tool. Available in `SDK_Folder\tools\GProgrammer`. |

## 5.2 Test and Verification

The DFU tests through UART include:

- DFU test by using GProgrammer

- DFU test by using DFU Master and DFU Boot firmware in a GR551x SDK

### 5.2.1 Test with GProgrammer

Download the firmware *ble_dfu_boot_fw.bin* to the GR5515 SK Board serving as the device (DFU Boot) through GProgrammer via UART and run the firmware. For details, see *GProgrammer User Manual*.

📖 **Note**:

The *ble_dfu_boot_fw.bin* is in `SDK_Folder\projects\ble\dfu\ble_dfu_boot\build\`.

### 5.2.2 Test with DFU Master and DFU Boot

For this test, GR5515 SK Board A serves as the host (DFU Master), and GR5515 SK Board B as the device (DFU Boot).

**📖 Note**:

The source code and project file of DFU Master are in `SDK_Folder\projects\ble\dfu\dfu_master`, and project file *dfu_master.uvprojx* is in the Keil_5 folder.

For the project directory of the DFU Boot firmware *ble_dfu_boot_fw.bin*, see "Section 5.2.1 Test with GProgrammer".

Steps for tests by using DFU Master and DFU Boot firmware in a GR551x SDK:

1.  Remove jumper caps of Pins 5–6 and Pins 7–8 at J5 pin header on the Board A and B.

2.  Connect Pin 5 at J5 on the Board A to Pin 7 at J5 on the Board B and Pin 7 at J5 on the Board A to Pin 5 at J5 on the Board B with DuPont wires, to establish connection between serial ports on the two boards.

3.  Download *dfu_master_fw.bin* to the Board A and *ble_dfu_boot_fw.bin* to the Board B with GProgrammer.

4.  Download the BIN file to be updated to the Board A with GProgrammer.

5.  Press appropriate keys on Board A to update firmware through UART according to tips on the display of Board A.

**🔔 Tip**:

- The load address and run address of *ble_dfu_boot_fw.bin* (DFU Boot) are 0x01002000.

- To prevent the downloaded DFU Boot firmware being overwritten, you need to set different load addresses for the firmware to be updated and the DFU Boot firmware. For details, see "Create Target Firmware for Update" in *GR551x OTA Example Application*.

# 6 FAQ

This chapter lists possible problems with corresponding solutions in DFU process for GR551x SoCs.

## 6.1 Why does DFU for Application Firmware on a Bare Device Fail?

- Description

  DFU through Bluetooth LE connection fails.

- Analysis

  There may be multiple reasons: For update through Bluetooth LE connection at 2.4 GHz with severe interference, the Bluetooth connection may be broken; improper configurations may also lead to update failure.

- Solution

  1. To update through Bluetooth LE connection, check the Bluetooth connection state of the device.

  2. To update through UART, check whether the baud rates of the host and of the device are consistent.

  3. Check whether the address of the firmware to be updated conflicts with the address of the current firmware.

## 6.2 Why do Exceptions Occur in OTA DFU for Application Firmware on a Device with an Operating System?

- Description

  DFU fails when a device uses FreeRTOS, leading to exceptions in other tasks or exceptions such as HardFault.

- Analysis

  The reason may be those described in "Section 6.1 Why does DFU for Application Firmware on a Bare Device Fail?"; it may also be because that a task stack with insufficient space has been assigned, leading to overflow of the task stack. The OTA task stack assigned shall be no less than 6 KB.

- Solution

  Assign a task stack no less than 6 KB to the OTA task.