BLE & HCI/UART Bootloader/DFU

Contents

nRF5 SDK v11.0.0-2.alpha

BLE & HCI/UART Bootloader/DFU

This information applies to the following SoftDevices: ***S130, S132***

The BLE Bootloader/DFU example projects demonstrate how to update the SoftDevice, bootloader, or application on an nRF5 SoC. This update process is called a Device Firmware Update, or DFU for short. The new SoftDevice, bootloader, or application image can be transferred over-the-air (OTA) using the Nordic BLE DFU Service.

The SDK provides several example projects that you can use as a starting point for developing your own DFU bootloader or application with DFU support. You can choose to use BLE transport (the default) or serial (HCI/UART) transport. If you want to use the ANT protocol instead, see Experimental: ANT Bootloader/DFU for documentation.

Note that if you program a DFU bootloader on the device, you must use this bootloader to install the application. Programming the application with other tools will not update the bootloader settings, which means that the application might not start. Erase the device if you do not want to use the DFU bootloader anymore.

The following sections describe the general concept of a DFU bootloader and explain how to use and adapt the BLE DFU bootloader examples that are included in the SDK:

- Architecture gives an overview of the components that are involved in a Device Firmware Update and how they interact, shows the general architecture, and presents the state machine that is the base for the implementation.
- Creating a DFU bootloader explains how to implement, program, and use a DFU bootloader.
- Adding DFU Service support to an application explains how to add DFU Service support to an application, so that the application can trigger the device to restart in DFU mode and perform a Device Firmware Update.
- Memory layout describes which areas of the device's memory are used to store which firmware.
- Transport layers gives detailed information about the transport protocols that can be used when performing a DFU.

For descriptions of the APIs and DFU procedures, see the Bootloader/DFU API section.

1. nRF5 SDK v11.0.0-2.alpha

Architecture

This information applies to the following SoftDevices: ***S130, S132***

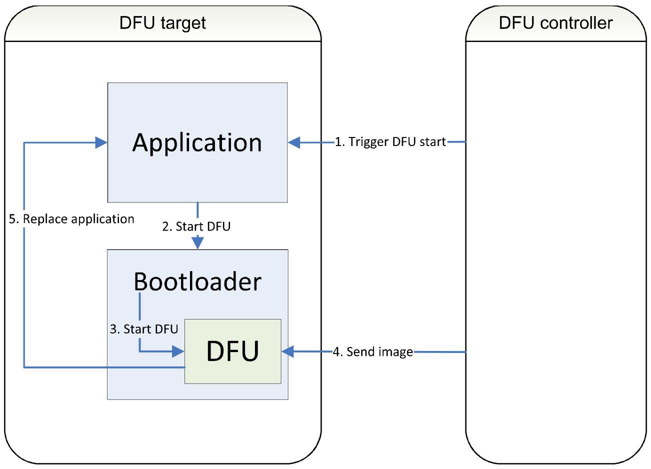
Two devices are required to perform a Device Firmware Update: the *DFU target* and the *DFU controller*. The DFU target is the device that is updated with a new firmware image, which can contain a new application, SoftDevice, bootloader, or a combination of SoftDevice and bootloader. The DFU controller is the device that transfers the image. For example, the DFU controller can be an app on a mobile phone, or an nRF5 dongle in conjunction with Master Control Panel.

The DFU target is the device that contains a bootloader with DFU capabilities. This bootloader's responsibility is to start either the application or the DFU mode, depending on different triggers. The DFU mode is part of the bootloader. If DFU mode is started, the bootloader expects a Device Firmware Update. The DFU controller can then initiate the transfer of a firmware image, which is received and validated by the bootloader. If the image is valid, it is copied to replace the existing firmware; depending on the type of image, it might replace the application, the SoftDevice, or even the current bootloader that is receiving the update.

By default, the DFU bootloader will start the application on the device. DFU mode is started in the following cases:

- No application is installed on the device.
- Button 4 is pressed when starting the device.
- The application on the DFU target supports entering DFU mode. In this case, the DFU controller can trigger the application to reset the device and enter DFU mode, which is referred to as *buttonless* update.

The following figure shows the workflow when performing a buttonless update of the application:



Example of a buttonless DFU to replace the application image

For a more technical overview, see the Architecture of the DFU bootloader and the Architecture of the DFU process.

The DFU state machine models the states and transitions in a DFU process.

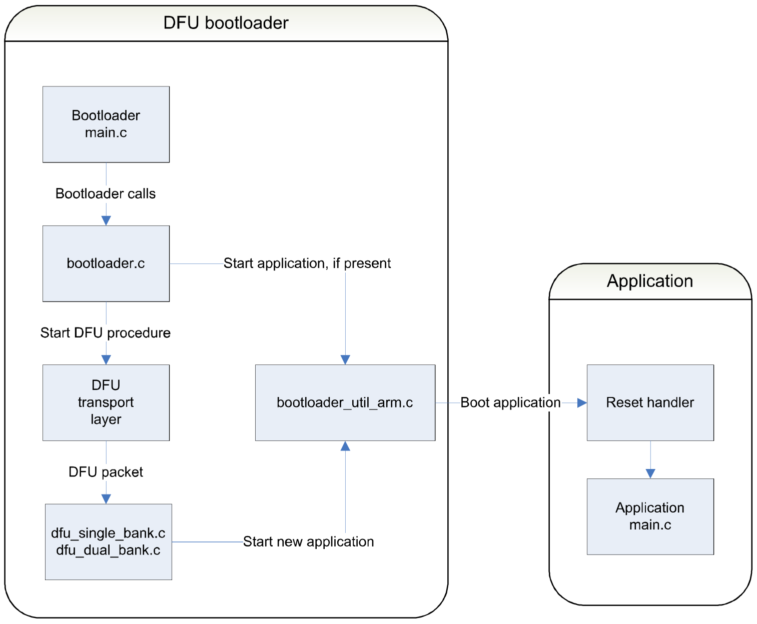
1.1. nRF5 SDK v11.0.0-2.alpha

Architecture of the DFU bootloader

This information applies to the following SoftDevices: ***S130, S132***

Unless triggered to start in bootloader mode, the DFU bootloader will check if a valid application is present on the device. If there is an application, the bootloader will run it. Otherwise, it will start the DFU procedure, receive DFU packets, and replace the existing firmware.

The following figure displays the blocks in the DFU bootloader and their tasks when performing an application update:



Architectural overview of the DFU bootloader

This document was last updated on Fri Dec 18 2015.
Please send us your feedback about the documentation! For technical questions, visit the Nordic Developer Zone.

1.2. nRF5 SDK v11.0.0-2.alpha

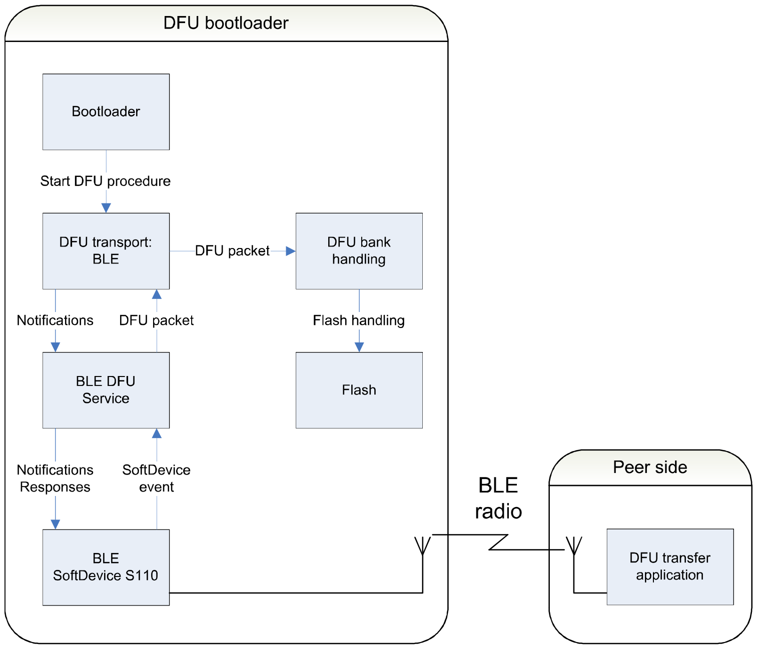
Architecture of the DFU process

*This information applies to the following SoftDevices: **S130, S132***

New firmware images can be transferred over two different transport protocols: BLE or using serial wire (HCI/UART). In both cases, the DFU bank handling is responsible for writing received data packets to flash memory.

The BLE transport uses a BLE service for data transfer (see BLE DFU Service) and relies on the S13x SoftDevice.

The following figure shows a schematic overview of the DFU process using BLE:

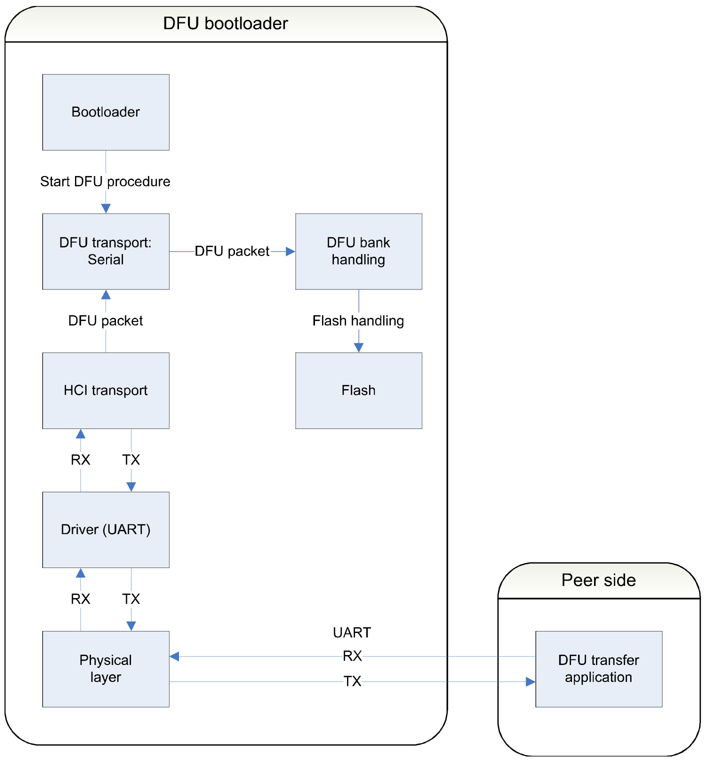


Architectural overview of the DFU process using the BLE transport protocol

Alternatively, new images can be transferred using HCI over UART (serial transport). The HCI transport layer increases robustness of the transfer. See HCI Transport for more information about UART_HCI, and Serial (HCI) packet format for more information about the packet format.

Serial transport does not use a SoftDevice. However, you must install a SoftDevice on your device to correctly create the Master Boot Record.

The following figure shows a schematic overview of the DFU process using with UART.



Architectural overview of the DFU process using serial transfer

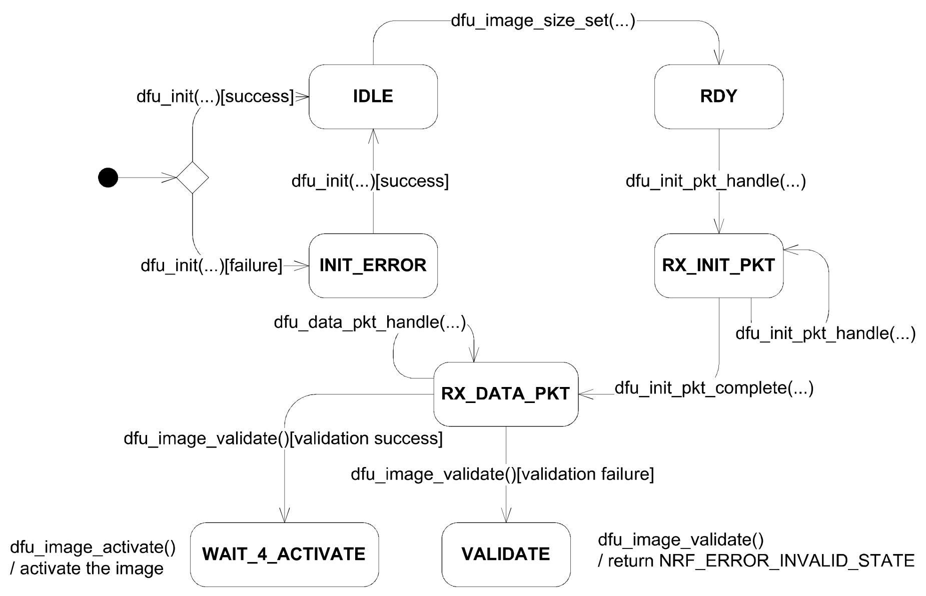
1.3. nRF5 SDK v11.0.0-2.alpha

DFU state machine

*This information applies to the following SoftDevices: **S130, S132***

The firmware update process has been modelled as an event-driven state machine, with each event signifying a specific DFU procedure. The events and procedures are agnostic of the transport protocol that is in use. Therefore, it is possible to perform a firmware update using any transport protocol. The DFU controller can trigger the events by sending a corresponding message on the DFU transport.

The following figure illustrates the transitions in the state machine:



State machine transitions in the DFU process

The entry point of the state machine is the bootloader starting up and entering DFU mode. During initialization, the selected transport layer is initialized. When an update process has been initiated, the DFU prepares the non-volatile memory region for the new application.

The following messages initiate transitions:

Message type	Description
dfu_init	Initialize the Device Firmware Update module.
dfu_image_size_set	Set the DFU image size.
dfu_init_pkt_handle	Receive the init packet to be used for safety-checking as descibed in Safety-checking the image.
dfu_init_pkt_complete	The complete init packet has been received; execute the safety check for the first step of the DFU process.
dfu_data_pkt_handle	Receive a portion of the new application.
dfu_image_validate	Trigger validation of the new application.
dfu_image_activate	Activate the transferred image after validation has successfully completed.
dfu_sys_reset	Reset the device. A reset can occur at any time, for example if a bootloader time-out occurs or when triggered by an external message.

2. nRF5 SDK v11.0.0-2.alpha

Creating a DFU bootloader

*This information applies to the following SoftDevices: **S130, S132***

The SDK provides some example projects that implement a bootloader with Device Firmware Update (DFU) capabilities, which can receive a firmware image and copy it to the nRF5 IC to replace the current bootloader, SoftDevice, or application.

The examples support the following Device Firmware Updates:

- Updating the SoftDevice.
- Updating the bootloader.
- Updating SoftDevice and bootloader in a single operation.
- Installing or updating an application.

Before updating the device firmware, make sure that the S130 or S132 SoftDevice v2.0.0-7.alpha or later is installed on the IC. For nRF51 ICs, the following prerequisites must be fulfilled as well:

- If the register UICR.CLENR0 has a size value (for example 0x00016000) and you are updating the SoftDevice, the new SoftDevice must not be larger than the current SoftDevice.
- When updating the SoftDevice, the bootloader, or SoftDevice and bootloader, the flash size of the target device must be 256 kB. When updating or installing an application, the flash size can be 128 kB or 256 kB.

If the device has an existing application when performing a SoftDevice update, the application is erased. Transferring a bootloader or application using dual-bank mode (see Dual-bank and single-bank updates) preserves the current image until the new image is copied, validated, and activated.

To get started, program and test one of the provided example projects. Running the BLE bootloader example describes the process for the default example, which uses BLE transport. Alternatively, you can run an example that uses serial transport (see Running the serial bootloader example).

The SDK provides sample images that you can use to test the DFU process. Creating an image file explains how you can convert your own image into the correct format.

The examples include a method for ensuring that the uploaded image is valid. See Safety-checking the image for information on how to ensure that only valid images are accepted.

If you encounter any errors, check the Troubleshooting section.

2.1. nRF5 SDK v11.0.0-2.alpha

Running the BLE bootloader example

*This example requires one of the following SoftDevices: **S130, S132***

Important: Before you run this example, make sure to program a SoftDevice.

Before you create your own DFU bootloader, run and test one of the provided DFU bootloader examples to understand the underlying concepts. Start with the basic example that uses BLE as transport protocol. See Running the serial bootloader example for information about the examples that use serial transport.

Setup

The name of the example is **dfu_dual_bank_ble_*SoftDevice_board***, where *SoftDevice* is the SoftDevice that you are using and *board* is the supported development board. If you are not using the Keil Pack Installer, you can find the source code and project file of the example in the following folder: `<InstallFolder>\examples\dfu\bootloader`

LED assignments:

- LED 1: Advertising
- LED 2: Connected
- LED 3: Bootloader mode active

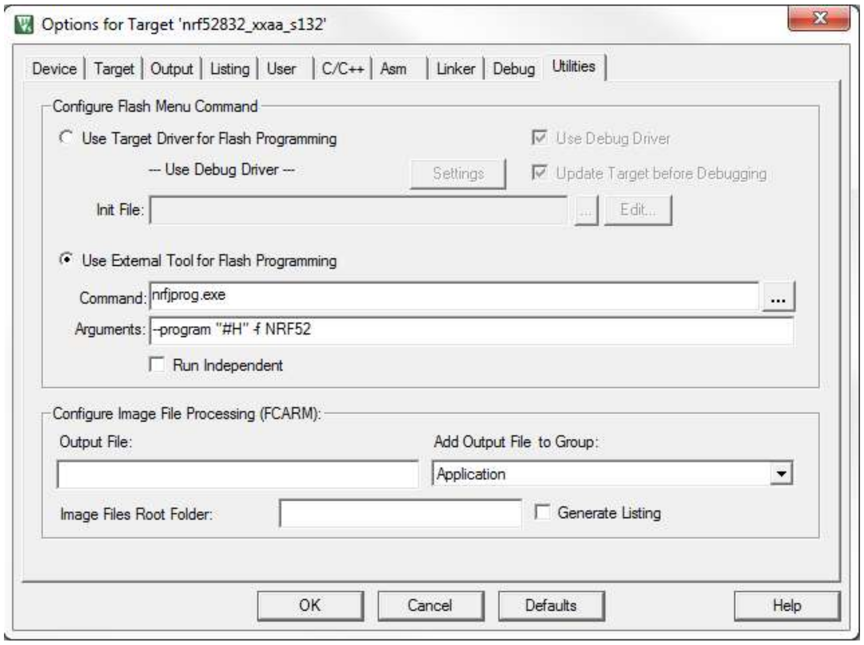
Button assignments:

- Button 4 (during startup): Enter bootloader mode

Programming the DFU bootloader

Complete the following steps to program the DFU bootloader example on your device using Keil:

1. Erase the device.
2. Program the SoftDevice. See Programming SoftDevices for instructions.
3. Compile the project **dfu_dual_bank_ble_*SoftDevice_board***.
4. Select **Project** > **Options for Target '*xxx*'** and make sure that `nrfjprog.exe` is configured as the tool for flash programming in the project options. The default J-Link target driver cannot correctly program the bootloader. The example project is preconfigured to use `nrfjprog.exe`, which is installed with the nRF5 MDK and must be in the Windows system path.
   The following screenshot shows the required settings for nrfjprog.exe (for nRF51 Series devices, change "NRF52" to "NRF51"):


Options for Target 'nrf52832_xxaa_s132'

Configure Flash Menu Command

Use Target Driver for Flash Programming

-- Use Debug Driver --

Use Debug Driver [✓]

Update Target before Debugging [✓]

Init File:

[✓ Use External Tool for Flash Programming]

Command: nrfjprog.exe

Arguments: --program "#H" -f NRF52

Run Independent [ ]

Configure Image File Processing (FCARM):

Output File:

Add Output File to Group: Application

Image Files Root Folder:

Generate Listing [ ]

Flash tool configuration in Keil

5. Program the DFU bootloader. If several J-Link emulators are connected, select the one that contains the nRF5 IC that you want to flash.

Updating the device firmware

Complete the following steps to perform a Device Firmware Update:

1. Create a firmware file or use one of the supplied example files:

Image	Description
dfu_test_bootloader_b.zip	Zip file containing a bootloader image and the corresponding init packet
dfu_test_softdevice_b.zip	Zip file containing a SoftDevice image and the corresponding init packet
dfu_test_softdevice_w_bootloader_b.zip	Zip file containing a combined bootloader and SoftDevice image and the corresponding init packet
dfu_test_app_hrm.zip	Zip file containing an application image and the corresponding init packet

All images are located in the `*BaseFolder*\examples\dfu\ble_dfu_send_hex\test_images_update_nrf5x` folder. If you are using Keil packs, the default `*BaseFolder*` is `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_Examples\*version*`. If you are using the repository distribution variant of the SDK, `*BaseFolder*` is `*InstallFolder*\examples`.
2. Make sure that you are using Master Control Panel v3.8 or later.
3. Start the device in bootloader mode. If there is no existing application on the device, the device will be started in bootloader mode automatically. Otherwise, press Button 4 during startup to put the device into bootloader mode. The advertising LED is lit.
4. Select the device in Master Control Panel. The device is advertising as "DfuTarg".
5. Connect to the device and perform service discovery. Observe that the connected LED is lit and the advertising LED is off. Master Control Panel will recognize that the device supports Device Firmware Update, and the "DFU" button in the user interface will become available.
   a. Click "DFU" and select the zip file that contains the firmware image.
   b. Click "Program". Observe that the update procedure starts. When the progress bar reaches 100%, the update is complete.

If an application was transferred, the application is started automatically. Observe that the Advertising LED is lit. If a bootloader or SoftDevice was transferred and no application is installed, the system will restart in bootloader mode.

Verifying the transferred firmware

If you used one of the provided example image files, you can verify that the Device Firmware Update completed as expected as follows:

Image	Verification
dfu_test_bootloader_b.zip dfu_test_softdevice_w_bootloader_b.zip	Open the Master Control Panel. The device is now advertising as "DfuTest" instead of "DfuTarg".

dfu_test_softdevice_b.zip dfu_test_softdevice_w_bootloader_b.zip	Check the data at location 0x00003000 by issuing the following command: **nrfjprog –memrd 0x3000 –w 32 –n 16** The data has changed to `FFFFFF10 51B1E5DB 0001B000 FFFF`**`FFFF`**. Before the update, the last two bytes were `0078` for the S130 SoftDevice 2.0.0-7.alpha or `0079` for the S132 SoftDevice 2.0.0-7.alpha (other SoftDevices might have a different value).
dfu_test_app_hrm.zip	Open the Master Control Panel. The device is now advertising as "Dfu_HRM" instead of "DfuTarg".

Advanced: IronPython script for DFU

Instead of using Master Control Panel to perform a Device Firmware Update, you can also use an IronPython script. In most cases, using the Master Control Panel is sufficient, but if you need or prefer a command line interface, you can use the IronPython script directly. The script is also used internally when you perform the DFU in Master Control Panel, so the provided functionality is the same.

The script requires IronPython and must be run on a Microsoft Windows operating system. By default, it is located in the `C:\Program Files (x86)\Nordic Semiconductor\Master Control Panel\<version>\lib\dfu\` folder.

Before you run the script, determine the device address of the device, which is advertising as "DfuTarg". You can find out the device address in Master Control Panel. However, make sure to close Master Control Panel before you run the script.

The script uses the following syntax:

```
ipy main.py --file <image_file.zip> --address <target_address>
```

- `<image_file.zip>` contains the firmware image and the init packet.
- `<target_address>` is the address of the device that is advertising as "DfuTarg".

This document was last updated on Fri Dec 18 2015.
Please send us your feedback about the documentation! For technical questions, visit the Nordic Developer Zone.

2.2. nRF5 SDK v11.0.0-2.alpha

Running the serial bootloader example

*This example requires one of the following SoftDevices: **S130, S132***

Important: Before you run this example, make sure to program a SoftDevice.

Instead of using BLE as transport protocol, you can also use serial transport. The SDK provides two example applications for serial (HCI) transport, one for a single-bank update and another for a dual-bank update. See Dual-bank and single-bank updates for more information about these update types.

The general setup and the process of programming the SoftDevice and bootloader are as described in Running the BLE bootloader example. However, you cannot use the Master Control Panel to perform a Device Firmware Update. Instead, nRFgo Studio provides a tool that is similar to the Advanced: IronPython script for DFU for BLE.

Updating the device firmware

Program the SoftDevice and bootloader as described in Programming the DFU bootloader, but use one of the examples for serial transport. The examples for serial transport do not actually require any SoftDevice functions, but installing the SoftDevice provides the Master Boot Record, which is required for running the bootloader.

The names for the examples are **dfu_single_bank_serial_*SoftDevice_board*** and **dfu_dual_bank_serial_*SoftDevice_board***. If you are not using Keil to work with the SDK, you can find the source code and project file of the examples in the following folder: `<InstallFolder>\examples\dfu\bootloader`

The tool that you use to update the device firmware is delivered with nRFgo Studio. To use it, ensure that you have installed nRFgo Studio v1.18 or later. By default, the script is located in the `C:\Program Files (x86)\Nordic Semiconductor\nRFgo Studio\` folder.

As test images, you can use the firmware images that are provided in the folders `<BaseFolder>\dfu\hci_dfu_send_hex\test_images_single_bank_update` or `test_images_dual_bank_update`, respectively. If you are using Keil packs, the default `<BaseFolder>` is `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_Examples\<version>`. If you are using the repository distribution variant of the SDK, `<BaseFolder>` is `<InstallFolder>\examples`. Use the images for the example that you are running. You cannot update a single-bank bootloader to a dual-bank bootloader or vice versa.

The script uses the following syntax:

```
nrfutil.exe dfu serial --package <image_file.zip> --port <com_port> [--flowcontr
ol] [--baudrate <baud_rate>]
```

- `<image_file.zip>` contains the firmware image and the init packet.
- `<com_port>` is the COM port to which the device is connected.
- `--flowcontrol` must be specified to enable flow control. If it is not specified, flow control is disabled. The provided examples use flow control, so you must specify this parameter.
- `<baud_rate>` is the desired baud rate. Valid values are 38400, 96000, 115200, 230400, 250000, 460800, 921600, and 1000000. Baud rates over 115200 are supported by the nRF5 IC, but might not be supported by all RS232 devices on Windows. If no baud rate is specified, the default baud rate 38400 is used. The provided examples use the default baud rate.

Use `nrfutil.exe --help` to display usage information.

Verifying the transferred firmware

If you used one of the provided example image files, you can verify that the Device Firmware Update completed as expected as follows:

Image	Verification
dfu_test_bootloader_b_hci.zip dfu_test_softdevice_w_bootloader_b_hci.zip	With the updated bootloader, LED 4 is lit instead of LED 3 when the device is in bootloader mode.
dfu_test_softdevice_b.zip dfu_test_softdevice_w_bootloader_b_hci.zip	Check the data at location 0x00003000 by issuing the following command: **nrfjprog –memrd 0x3000 –w 32 –n 16** The data has changed to `FFFFFF10 51B1E5DB 0001B000 FFFF`**`FFFF`**. Before the update, the last two bytes were `0078` for the S130 SoftDevice 2.0.0-7.alpha or `0079` for the S132 SoftDevice 2.0.0-7.alpha (other SoftDevices might have a different value).
dfu_test_app_hrm.zip	Open the Master Control Panel. The device is advertising as "Dfu_HRM".

2.3. nRF5 SDK v11.0.0-2.alpha

Creating an image file

*This information applies to the following SoftDevices: **S130, S132***

The format of the firmware image that you use to update the device firmware depends on the tool that you use to perform the DFU. The DFU bootloader expects the image in binary format; however, some tools automatically convert the image from HEX format to binary format.

Creating a zip with image and init packet

If you use Master Control Panel or other tools by Nordic Semiconductor to update the device firmware, you must provide a zip that contains the image file and a corresponding init packet. To create this zip file, use the `nrfutil.exe` tool that is installed with the Master Control Panel (named `nrf.exe` in Master Control Panel versions <3.10.0). By default, the tool is located in the `C:\Program Files (x86)\Nordic Semiconductor\Master Control Panel\<version>\nrf\` folder on Windows. For Linux and OS X users, nrfutils is available as a standalone python package with command line utility on the Nordic Semiconductor GitHub profile. Run `nrfutil.exe dfu genpkg --help` to display usage instructions.

You can add the following firmware images in binary format to the zip file:

- `--application` *image.bin:* a binary image of an application
- `--bootloader` *image.bin:* a binary image of a bootloader
- `--softdevice` *image.bin:* a binary image of a SoftDevice

You can also combine several images in one zip file.

In addition to the images, you must specify the information that will be added to the init packet:

- `--application-version` *version:* the version of the application image, for example, `0xff`
- `--dev-revision` *version:* the revision of the device that should accept the image, for example, `1`
- `--dev-type` *type:* the type of the device that should accept the image, for example, `1`

- `--sd-req` *sd_list:* a comma-separated list of FWID values of SoftDevices that are valid to be used with the new image, for example, `0x4f,0x5a`

Creating a binary image

When you compile an application in Keil, two images are created:

- A HEX file
- An AXF (ELF) file

In most cases, you can use the image in HEX format to perform the DFU. If you need a binary image, use the command line tool `fromelf.exe` to convert the AXF file into a binary image. The tool is located in the `<KeilFolder>\ARM\ARMCC\bin` folder.

Call `fromelf.exe` as follows to convert an AXF image into a binary file:

```
<Keil-folder>\ARM\ARMCC\bin\fromelf.exe --bin --output <outfile.bin> <infile.axf
>
```

2.4. nRF5 SDK v11.0.0-2.alpha

Safety-checking the image

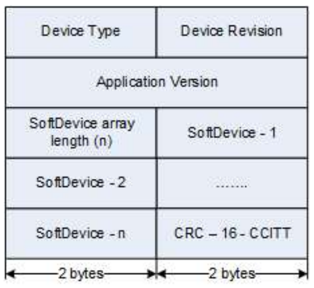
*This information applies to the following SoftDevices: **S130, S132***

To make sure that only compatible applications are installed on a device, the DFU procedure includes a mechanism to safety-check the transferred firmware image.

When updating the application on the nRF5 IC, the image that is transferred must be accompanied by an init packet that contains information about the image. The tool that you use to perform the DFU must send this packet before transferring the actual image. The DFU processing in the bootloader must check the information in the init packet to ensure that the transferred image is valid and to accept only compatible applications.

The init packet contains the following information that is used for safety checks (see dfu_init_packet_t):

- Device type: A 2-byte value specified by the developer that identifies the device type, for example Heart Rate Belt.
- Device revision: A 2-byte value that can be used to restrict the update to be accepted only on devices with a defined revision number.
- Application version: A 4-byte value identifying the version of the application that is being transferred. This value can be used to allow only software upgrades and prevent downgrades. No example code is provided for this feature.
- Supported SoftDevices: A list of 2-byte values identifying the SoftDevices that are compatible with the application, for example, S132 v2.0.0.
- Checksum: A 2-byte CRC-16-CCITT for the image to transfer.



DFU init packet

Sending the init packet

How to send the init packet depends on the procedure that you use to perform the DFU.

If you use Master Control Panel or a Python script to perform the update, you must provide a zip file that contains the image and the init packet.

If you use Nordic Semiconductor's Android or iOS app to perform the update, the required file format depends on the version of the app. New versions support zip files that contain the firmware image and the init packet. Older versions support BIN or HEX files and will prompt you to select an init packet to send.

Checking the init packet

The nRF5 SDK provides a template, `dfu_init_template.c`, to perform safety checks of the init packet. The template is located in the `<BaseFolder>\bootloader_dfu` folder. If you are using Keil packs, the default `<BaseFolder>` is `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_Libraries\<version>`. If you are using the repository distribution variant of the SDK, `<BaseFolder>` is `<InstallFolder>\components\libraries`.

`dfu_init_template.c`, which is also used in the DFU bootloader example projects, can be used as a starting point to develop procedures that increase the safety of the DFU. The current implementation includes checks for Device type and revision, Supported SoftDevices, and the checksum, but not for the Application version.

Device type and revision

The device type and revision are stored in the user-reserved area of UICR (0x10001080) on the nRF5 IC. If this location is used for other purposes, update the offset UICR_CUSTOMER_DEVICE_INFO_OFFSET in `dfu_init.h` to match a free location in UICR.

```
#define UICR_CUSTOMER_DEVICE_INFO_OFFSET    0x0                                 /**<
      Device info offset inside the customer UICR reserved area. Customers may change this value
      to place the device information in a user-preferred location. */
```

The values stored at this location are compared to the values from the init packet. If they match, the image is accepted. Otherwise, the image is rejected. To accept all device types and revisions and to disable the check, make sure not to set the UICR value to a specific value, but keep the default value 0xFFFF.

Application version

`dfu_init_template.c` does not check the application version. However, you should implement an application version check if required.

If you add an application version check, every application must be compiled with a version ID. This version ID can be placed at a predefined location in the application image, for example at the application start address + 0x0100, similar to the principle used by Nordic Semiconductor's SoftDevices.

See the following code snippet from `dfu_init_template.c`, which illustrates where to extend the DFU Init packet handling with an application version safety check:

```
    // To support application versioning, this check should be updated.
    // This template allows for any application to be installed. However,
    // customers can place a revision number at the bottom of the application
    // to be verified by the bootloader. This can be done at a location
    // relative to the application, for example the application start
    // address + 0x0100.
```

Supported SoftDevices

Applications that are compiled for the nRF5 IC target a specific SoftDevice, for example S132 v2.0.0. Some applications might work with multiple SoftDevice versions if the API is backward compatible. For example, an application that is compiled for S132 SoftDevice v1.0.0 can also run on S132 SoftDevice v2.0.0.

Provide a list of supported SoftDevices for the application that is to be installed in the DFU init packet. The DFU procedure in the bootloader checks the list that is provided in the init packet against the currently installed SoftDevice on the IC and continues the update procedure only if a matching SoftDevice is installed.

Use a value of 0xFFFE in the init packet if the application should be installed regardless of the SoftDevice that is present. This feature can be helpful during development, but you should not use it in a product.

See the following table for the FWID values of current SoftDevices:

SoftDevice	FWID
S130 v2.0.0-7.alpha	0x0078
S132 v2.0.0-7.alpha	0x0079
Development/any	0xFFFE

2.5. nRF5 SDK v11.0.0-2.alpha

Troubleshooting

*This information applies to the following SoftDevices: **S130, S132***

The following sections describe some common errors that you might encounter when creating a DFU bootloader and their solutions.

No Algorithm found for: 10001014H - 10001017H

The following error message is displayed in Keil when programming the DFU bootloader:

```
No Algorithm found for: 10001014H - 10001017H
Partial Erase Done (areas with no algorithms skipped!)
No Algorithm found for: 10001014H - 10001017H
```

Cause: You attempted to use the default J-Link target driver to flash the DFU bootloader to the device. This does not work, because in addition to flashing the bootloader, the bootloader start address must be written to the UICR register. The default J-Link driver cannot do this.

Solution: Use the external tool `nrfjprog.exe` instead of the default J-Link driver. You can configure the tool that is used for flash programming on the Utilities tab of the project options in Keil.

Failed to execute "nrfjprog.exe"

The following error message is displayed in Keil when programming the DFU bootloader:

```
Error: failed to execute "nrfjprog.exe"
```

Cause: The external tool `nrfjprog.exe` that is required to flash the DFU bootloader to the device cannot be found in the Windows PATH variable.

Solution: Add the path to `nrfjprog.exe` to the Windows PATH variable, or add the full path to `nrfjprog.exe` on the Utilities tab of the project options in Keil. The tool is delivered with the nRF5 MDK, which is required to use the SDK. The default path to `nrfjprog.exe` is `C:\Program Files (x86)\Nordic Semiconductor\nrf51\bin\nrfjprog.exe`.

Error L6406E: No space in execution regions

The following error message is displayed in Keil when compiling the DFU bootloader:

```
linking...
.\_build\nrf51422_xxac.axf: Error: L6406E: No space in
execution regions with .ANY selector matching \c .....
```

Cause: The size of the compiled bootloader project exceeds the size that is configured in the project settings. This problem can occur if you modify the example code so that the resulting image becomes much bigger, or if you modify the project settings that define the memory areas on the IC.

Solution: Try using a higher optimization level (`-O3`), or increase the code area for the DFU bootloader as described in Relocating the bootloader.

DFU bootloader stops working after changing the start address

After changing the start address of the bootloader and flashing the image to the device again, the bootloader stops working.

Cause: Flashing the image to the device at a different memory location does not correctly set the pointer to the start address that is stored in the UICR register. Therefore, the pointer might point to the wrong start address, and there is no bootloader at this location.

Solution: Erase the device. This will delete the start address that is stored in the UICR register. Then program the SoftDevice and the DFU bootloader again. See Relocating the bootloader for more information.

Uploaded application does not run

You uploaded an application, but the application does not run when the device is reset.

Cause: The application cannot be started because it does not start at the expected start address.

Solution: Compile the application with the correct start address. For example, the correct start address for the S13x SoftDevice v2.0.0-7.alpha is 0x0001B000. To determine the correct start address for the SoftDevice you are using, check the SoftDevice Specification (SDS). See Memory layout for more information.

Access to the port 'COMxx' is denied

When performing a Device Firmware Update using the IronPython script, the following error is reported:

```
[EXCEPTION] Access to the port 'COMxx' is denied.
```

Cause: The script cannot access the device at the specified address, either because you specified an incorrect address or because the device is in use by another application.

Solution: Make sure that the specified address is correct and close other applications that might use the device, for example, the Master Control Panel.

3. nRF5 SDK v11.0.0-2.alpha

Adding DFU Service support to an application

*This information applies to the following SoftDevices: **S130, S132***

Instead of entering bootloader mode by pressing a button, you can also integrate BLE DFU Service support into an application and enable the application to automatically enter bootloader mode. Writing to the DFU Service will then trigger the application to restart in bootloader mode.

The general concept of the Device Firmware Update is the same no matter if you use a button to enter bootloader mode or add BLE DFU Service support to the application. In both cases, a DFU bootloader is required, and the DFU is performed when the device is in bootloader mode. The only difference is how the device is put into bootloader mode.

Adding BLE DFU Service support to an application is required whenever you want to be able to start bootloader mode and update a device without physical interaction. Potential reasons for this are, for example, that the device does not have a button or is not in reach.

To get started, program and test the provided Example application. Then check Extending your application for detailed information on how to include BLE DFU Service support in your own application. Switching to bootloader/DFU mode describes how to trigger bootloader/DFU mode.

The example application supports Sharing bonding information, which enables the DFU target to do directed advertising after bootloader mode is started, so that only the DFU controller can reconnect and the link is immediately re-encrypted.

3.1. nRF5 SDK v11.0.0-2.alpha

Example application

*This information applies to the following SoftDevices: **S130, S132***

To show how to add BLE DFU Service support to an application, the SDK contains a version of the Heart Rate Application that includes DFU support. This example illustrates how a standard application can be extended so that you can easily update the device firmware at any time, just by connecting to the application.

The name of the example is **ble_app_hrs_*SoftDevice*_with_dfu_*board***, where *board* is the supported development board. If you are not using the Keil Pack Installer, you can find the source code and project file of the example in the following folder: `<InstallFolder>\examples\ble_peripheral\ble_app_hrs`

When testing the example, you can choose to use bonding to share encryption keys between the application and the bootloader. If you use bonding, the link between the DFU target and the DFU controller will be re-established with encryption when the device restarts in bootloader mode. See Sharing bonding information for more information.

Testing the example

Test the Heart Rate Application with BLE DFU Service support with the Master Control Panel by performing the following steps:

1. Program the DFU bootloader.
2. Compile the application and perform a Device Firmware Update to program it. Observe that the BSP_INDICATE_ADVERTISING state is indicated.
3. Connect to the device from Master Control Panel, and optionally bond to it.
4. Perform service discovery. The device is advertising as "Nordic_HRM". Observe that the BSP_INDICATE_CONNECTED state is indicated.
5. Click "Enable services".
6. Click "DFU" to start the Device Firmware Update (see step 5 in Updating the device firmware).

Testing the example manually (with bonding)

Test the Heart Rate Application with BLE DFU Service support using bonding with the Master Control Panel by performing the following steps:

1. Program the DFU bootloader.
2. Compile the application and perform a Device Firmware Update to program it. Observe that the BSP_INDICATE_ADVERTISING state is indicated.
3. Connect to the device from Master Control Panel, then bond to it.
4. Perform service discovery. The device is advertising as "Nordic_HRM". Observe that the BSP_INDICATE_CONNECTED state is indicated.
5. Write the value "0200" to the Service Changed characteristic to enable the Service Changed indication when the GATT Server Attribute Table changes.
6. Write the value "0100" to the CCCD for the DFU Control Point characteristic or click Enable services to enable DFU notifications.
7. Write the command to start a DFU procedure to the DFU Control Point:
   - "01-01" if you want to update the SoftDevice
   - "01-02" if you want to update the bootloader
   - "01-03" if you want to update the SoftDevice and bootloader
   - "01-04" if you want to update the application
8. Observe that the connection to the device is lost. The message "SERVER: Received Link Loss" is displayed in the log window.
9. The device is now performing directed advertising to the bonded peer. When the connection is established, a Service Changed indication is received and displayed in the Master Control Panel.
10. Perform a Device Firmware Update. You do not need to connect to the device again, because you are already connected.

Testing the example manually (without bonding)

Test the Heart Rate Application with BLE DFU Service support not using bonding with the Master Control Panel by performing the following steps:

1. Program the DFU bootloader.
2. Compile the application and perform a Device Firmware Update to program it. Observe that the BSP_INDICATE_ADVERTISING state is indicated.
3. Connect to the device from Master Control Panel.
4. Perform service discovery. The device is advertising as "Nordic_HRM". Observe that the BSP_INDICATE_CONNECTED state is indicated.
5. Write the value "0100" to the CCCD for the DFU Control Point characteristic or click Enable services to enable DFU notifications.
6. Write the command to start a DFU procedure to the DFU Control Point:
   - "01-01" if you want to update the SoftDevice
   - "01-02" if you want to update the bootloader
   - "01-03" if you want to update the SoftDevice and bootloader
   - "01-04" if you want to update the application
7. Observe that the connection to the device is lost. The message "SERVER: Received Link Loss" is displayed in the log window.
8. Click "Back" in the Master Control Panel to see an overview of the discovered devices.
9. Clear the discovered devices and click "Start discovery".
10. Verify that the device is now advertising as "DfuTarg".
11. Perform a Device Firmware Update.

Extending your application

*This information applies to the following SoftDevices: **S130, S132***

You can extend any BLE application to support the BLE DFU Service, so that you can easily enter bootloader mode to update the device firmware at any time.

The following changes are required to add BLE DFU Service support to an application:

- The application must use the Device Manager. If the application does not use the Device Manager yet, you must update the application. See BLE Device Manager for more information.
- You must add DFU related files to the BLE application project. See Including DFU files in application.
- You must implement a function that executes application-specific operations to ensure a graceful shutdown of the application. See Implementing graceful shutdown.
- The application must correctly set up the BLE services. See Setting up the BLE services.
- The application must propagate SoftDevice BLE events to the BLE DFU Service. See Propagating BLE stack events.

With these changes to the application, the application can restart the device in bootloader mode to accept firmware updates. To remotely trigger the application to restart the device in bootloader mode, you must write "DFU Start" to the DFU Service Control Point. See Switching to bootloader/DFU mode.

Including DFU files in application

Perform the following steps to include the files that are required to support the BLE DFU Service in an application:

1. Open the application project in Keil.
2. Click **Project** > **Manage** > **Components, Environment, Books** or the respective icon.
3. On the Project Items tab, add a group with the name "Dfu".
4. Select the new group and click **Add Files**.
5. Add the following files:
   - **bootloader_util_arm.c**
     (Located in `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_Libraries\<version>\bootloader_dfu` if you are using Keil packs and `<InstallFolder>\components\libraries\bootloader_dfu` if you are using the repository distribution variant of the SDK.)
   - **dfu_app_handler.c**
     (Located in `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_Libraries\<version>\bootloader_dfu` if you are using Keil packs and `<InstallFolder>\components\libraries\bootloader_dfu` if you are using the repository distribution variant of the SDK.)
   - **ble_dfu.c**
     (Located in `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_BLE\<version>\ble_services\ble_dfu` if you are using Keil packs and `<InstallFolder>\components\ble\ble_services\ble_dfu` if you are using the repository distribution variant of the SDK.)

Implementing graceful shutdown

Before the application switches to bootloader mode, it should shut down gracefully. To achieve that, you must implement a function that is called before the shutdown.

`dfu_app_handler.c` supports a reset_prepare callback function, which will be executed before the reset and allows the application to perform any tasks that are considered important before restarting in bootloader mode. For example:

- Gracefully disconnect and close any active BLE links
- Wait for pending flash operation to finish to ensure known state
- Clean up registers
- Turn off LEDs

The reset will not occur until the reset_prepare function returns.

See the following example of a reset_prepare function from `main.c` of the Heart Rate Application with BLE DFU Service support:

```
/**@brief Function for preparing for system reset.
 *
 * @details This function implements @ref dfu_app_reset_prepare_t. It will be called by
 *          @ref dfu_app_handler.c before entering the bootloader/DFU.
 *          This allows the current running application to shut down gracefully.
 */
static void reset_prepare(void)
{
    uint32_t err_code;

    if (m_conn_handle != BLE_CONN_HANDLE_INVALID)
    {
        // Disconnect from peer.
        err_code = sd_ble_gap_disconnect(m_conn_handle,
      BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
        APP_ERROR_CHECK(err_code);
        err_code = bsp_indication_set(BSP_INDICATE_IDLE);
        APP_ERROR_CHECK(err_code);
    }
    else
    {
        // If not connected, the device will be advertising. Hence stop the advertising.
        advertising_stop();
    }

    err_code = ble_conn_params_stop();
    APP_ERROR_CHECK(err_code);

    nrf_delay_ms(500);
}
```

Setting up the BLE services

To be able to trigger an update remotely, you must initialize the BLE DFU Service and enable the Service Changed characteristic. In addition, you must make sure that the Device Manager knows about the new Service Changed characteristics.

Initializing the DFU Service

The BLE DFU Service is required to connect to the application and trigger a restart into bootloader mode. To be able to write to the BLE DFU Service, you must initialize the service in the application.

See the following example code from `main.c` of the Heart Rate Application with BLE DFU Service support:

```
#include "ble_dfu.h"
#include "dfu_app_handler.h"
```

```
    ble_dfu_init_t   dfus_init;

    // Initialize the Device Firmware Update Service.
    memset(&dfus_init, 0, sizeof(dfus_init));

    dfus_init.evt_handler   = dfu_app_on_dfu_evt;
    dfus_init.error_handler = NULL;
    dfus_init.evt_handler   = dfu_app_on_dfu_evt;
    dfus_init.revision      = DFU_REVISION;

    err_code = ble_dfu_init(&m_dfus, &dfus_init);
    APP_ERROR_CHECK(err_code);

    dfu_app_reset_prepare_set(reset_prepare);
    dfu_app_dm_appl_instance_set(m_app_handle);
```

Enabling the Service Changed characteristic

You must enable the Service Changed characteristic so that changes in the application are indicated to other devices, most importantly the DFU controller.

To enable the characteristic, simply set IS_SRVC_CHANGED_CHARACT_PRESENT to 1 in the application:

```
#define IS_SRVC_CHANGED_CHARACT_PRESENT    0
```

Updating the Device Manager configuration

The Device Manager keeps track of the characteristics that are enabled for an application. Therefore, you must update the Device Manager configuration to account for the two new Service Changed characteristics (one for the application itself and one for the DFU Service).

Set the maximum number of characteristic client descriptors in the file `device_manager_cnfg.h` (located in `C:\Keil\ARM\Pack\NordicSemiconductor\nRF_BLE\<version>\device_manager\config` if you are using Keil packs and `<InstallFolder>\components\ble\device_manager\config` if you are using the repository distribution variant of the SDK):

```
#define DM_GATT_CCCD_COUNT                    4
```

Propagating BLE stack events

The BLE DFU Service relies on BLE stack events. Therefore, for the service to function properly, all BLE Stack events from the SoftDevice must be propagated to the DFU Service.

To propagate BLE stack events to the BLE DFU Service, call ble_dfu_on_ble_evt with the BLE stack event as a parameter. In most SDK examples, BLE stack events are propagated to submodules in the function ble_evt_dispatch(ble_evt_t * p_ble_evt) in `main.c`.

See the following example code from `main.c` of the Heart Rate Application with BLE DFU Service support:

```
    ble_dfu_on_ble_evt(&m_dfus, p_ble_evt);
```

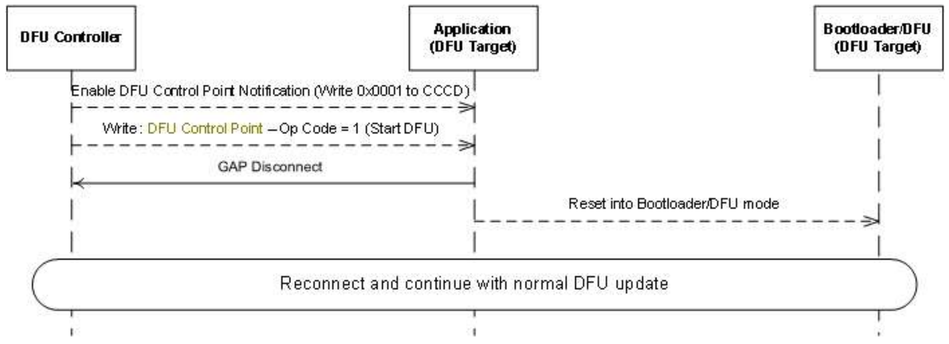
3.3. nRF5 SDK v11.0.0-2.alpha

Switching to bootloader/DFU mode

*This information applies to the following SoftDevices: **S130, S132***

After programming both DFU bootloader and application on the device and starting the application, you can switch to bootloader/DFU mode at any time by writing to the control point of the DFU Service.

The following flow chart shows the process of switching from a BLE application to bootloader/DFU mode:


DFU Controller
Application (DFU Target)
Bootloader/DFU (DFU Target)
Enable DFU Control Point Notification (Write 0x0001 to CCCD)
Write : DFU Control Point – Op Code = 1 (Start DFU)
GAP Disconnect
Reset into Bootloader/DFU mode
Reconnect and continue with normal DFU update

Switching from running an application to bootloader/DFU mode

First, you must write the value 0100 to the CCCD for the DFU Control Point characteristic to enable DFU notifications. Next, you must write the value 01-*XX* (where *XX* is the code for the type of image, see DFU Control Point) to the DFU Control Point to start a DFU procedure. The application will then disconnect and reset into bootloader/DFU mode. Note that no acknowledgement will be sent when writing to the Control Point, but the application sends a GAP disconnect on the link to tell the peer that the link is closed. You can then transfer the firmware image.

For more information about the message format and available commands, see DFU Control Point.

3.4. nRF5 SDK v11.0.0-2.alpha

Sharing bonding information

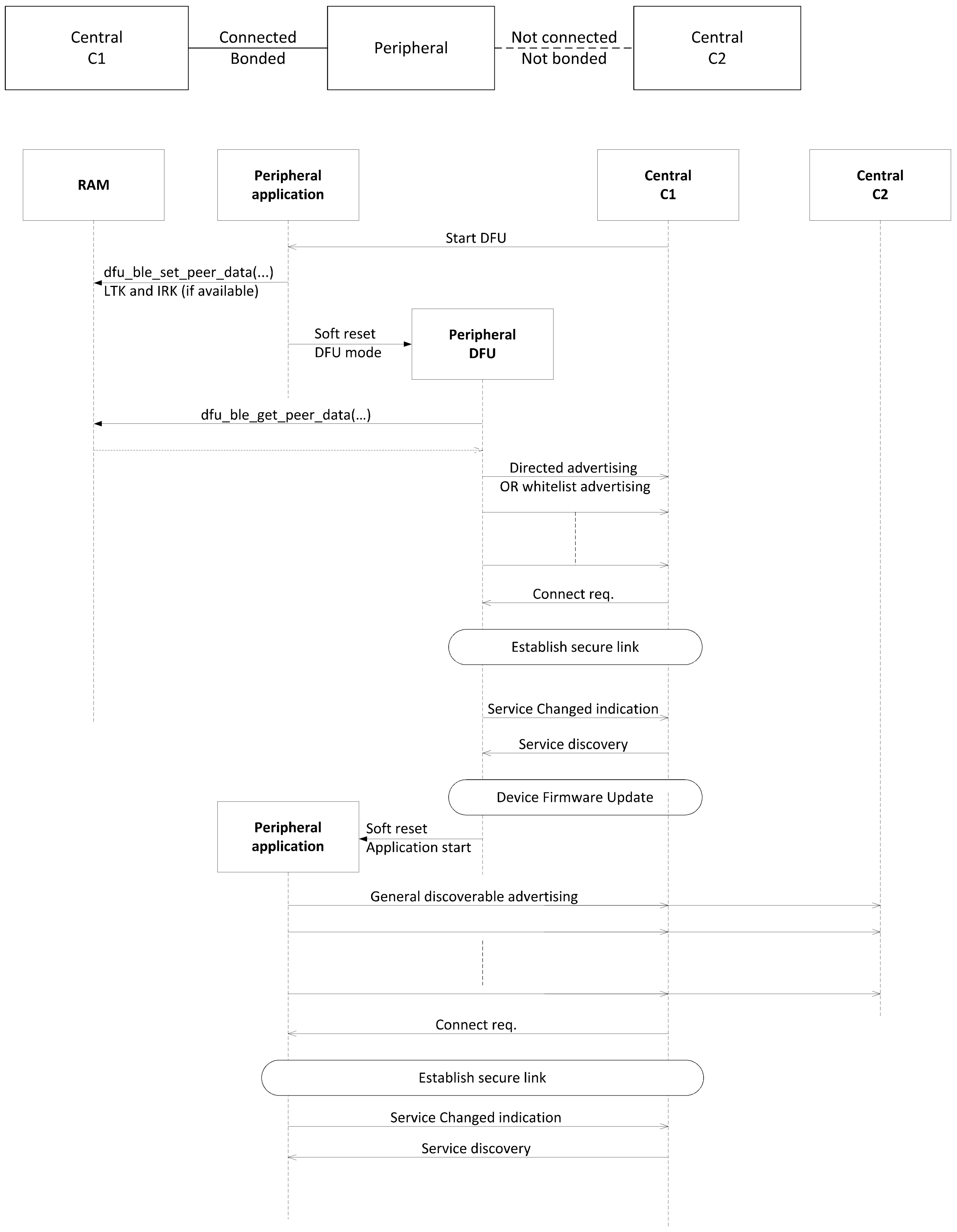
*This information applies to the following SoftDevices: **S130, S132***

When BLE DFU Service support is added to an application that uses the Device Manager, it is possible to transfer bonding information from the application to the bootloader. If the encryption keys are shared, both the application and the bootloader can reconnect to bonded devices. If no bonding information is available, the connection is re-established using directed advertising.

Bonding information consists of the peer address, the Identity Resolving Key (IRK), and the Long Term Key (LTK) of the connected peer.

If you include the file `dfu_app_handler.c` in your application, bonding information is automatically shared, and either directed advertising or whitelist advertising is used to re-establish the secure connection between the devices. Whitelist advertising is used if an Identity Resolving Key (IRK) has been distributed. In this case, the IRK of only this device is added to the whitelist, so that the device will allow connections only from the device that triggered the DFU mode. If the peer does not reconnect to the device, the system resets and loads the installed application again.

The following figure shows the process of reconnecting to a bonded device:



DFU on a bonded device

In bootloader mode, the device is advertising with the same address as in application mode if the application supports Service Changed indications. Otherwise, changes to the application cannot be indicated, and therefore the device must advertise as a new device. This is accomplished by the device advertising with the address increased by 1.

Implementation

The example bootloader uses a particular region of memory to pass the bonding information from the application to the bootloader. This memory region is designated at compile time not to be zero-initialized during a soft reset. To preserve the bonding information, the application performs a supervisor call (SVC) to the bootloader. The bootloader's SVC handler then copies the bonding information into the preserved memory region, so that the bootloader can read it back after the reset. A CRC is used to verify the integrity of the information.

Implementation in the application

In the example application, the keys that are needed to re-establish the secure connection are retrieved from the Device Manager using the function dm_distributed_keys_get. They are then passed from the application to the bootloader. The following code from `dfu_app_handler.c` shows how to implement bond sharing:

```
    err_code = dm_handle_get(conn_handle, &m_dm_handle);
    if (err_code == NRF_SUCCESS)
    {
        err_code = dm_distributed_keys_get(&m_dm_handle, &key_set);
        if (err_code == NRF_SUCCESS)
        {
            APP_ERROR_CHECK(err_code);

            m_peer_data.addr              = key_set.keys_central.p_id_key->id_addr_info;
            m_peer_data.irk               = key_set.keys_central.p_id_key->id_info;
            m_peer_data.enc_key.enc_info  = key_set.keys_periph.enc_key.p_enc_key->enc_info;
            m_peer_data.enc_key.master_id = key_set.keys_periph.enc_key.p_enc_key->master_id;

            err_code = dfu_ble_svc_peer_data_set(&m_peer_data);
            APP_ERROR_CHECK(err_code);

            app_context_data   = (DFU_APP_ATT_TABLE_CHANGED << DFU_APP_ATT_TABLE_POS);
            app_context.len    = sizeof(app_context_data);
            app_context.p_data = (uint8_t *)&app_context_data;
            app_context.flags  = 0;

            err_code = dm_application_context_set(&m_dm_handle, &app_context);
            APP_ERROR_CHECK(err_code);
        }
        else
        {
            // Keys were not available, thus we have a non-encrypted connection.
            err_code = dm_peer_addr_get(&m_dm_handle, &m_peer_data.addr);
            APP_ERROR_CHECK(err_code);

            err_code = dfu_ble_svc_peer_data_set(&m_peer_data);
            APP_ERROR_CHECK(err_code);
        }
    }
```

The dfu_ble_svc_peer_data_set function that is called in this example is a supervisor call to the bootloader. This means that the function is implemented in the bootloader code, but called from the application. For this SVC to be handled correctly, the application must know where to find the bootloader's SVC handler. To define its location, the application must call sd_softdevice_vector_table_base_set with NRF_UICR->NRFFW[0] as argument. See `dfu_app_handler.c` for the implementation in the example application.

Note

If sd_softdevice_vector_table_base_set is not called with NRF_UICR->NRFFW[0] as argument, the application will deadlock or behave erratically when dfu_ble_svc_peer_data_set is called.

Implementation in the bootloader

To ensure that the supervisor call is correctly handled by the bootloader's SVC handler and not the application's SVC handler, the bootloader must implement an SVC handler. In the example bootloader, the SVC handler (and also the dfu_ble_svc_peer_data_set function) is implemented in the file `dfu_ble_svc.c`.

Note

Make sure to implement an SVC handler in the bootloader. If no SVC handler is defined in the bootloader, the application will deadlock when dfu_ble_svc_peer_data_set is called.

4. nRF5 SDK v11.0.0-2.alpha

Memory layout

*This information applies to the following SoftDevices: **S130, S132***

When implementing a mechanism to perform a Device Firmware Update, you must be aware of where in the device memory the different firmware components are located.

During system start-up, the Master Boot Record will determine the start address of the bootloader by checking the address that is defined at UICR.NRFFW[0] (which can also be accessed by UICR.BOOTLOADERADDR on nRF51 devices). See Relocating the bootloader for information on how to change the start address of the bootloader.

The MBR will then initiate the bootloader. During initialization, the bootloader will either enter DFU mode to perform a firmware update or request the SoftDevice to start the application. The application is located right after the SoftDevice, so the SoftDevice is aware of its start address.

Firmware updates can be performed as dual-bank or single-bank updates. See Dual-bank and single-bank updates for more information and a detailed description of the update process.

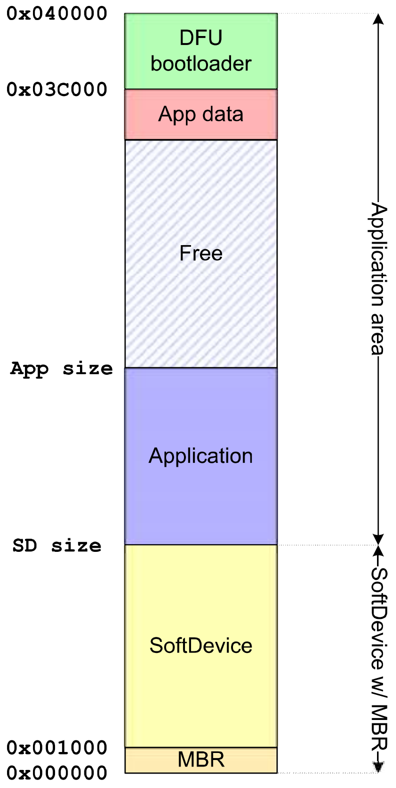
Application data can be preserved during an update. See Preserving application data for instructions.

Default memory layout

SoftDevices have different sizes. For example, the S13x SoftDevice v2.0.0-7.alpha ends at 0x0001B000.

nRF51 Series devices

The following figure displays the default memory layout of a 256 kB nRF51 device:



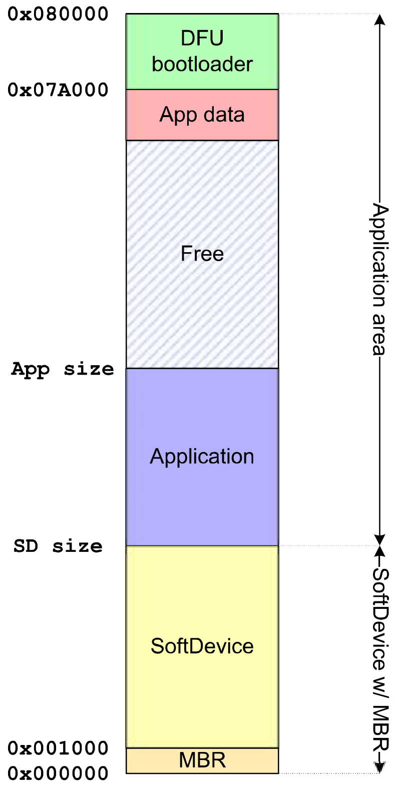
Memory layout (256 kB)

The following table shows the memory layout on a 256 kB nRF51 device with the S130 SoftDevice v2.0.0-7.alpha:

Memory range	Usage
0x0003C000 - 0x00040000	DFU Bootloader and data
0x0001B000 - 0x0003C000	Application code, free/swap, and application data
0x00001000 - 0x0001B000	SoftDevice
0x00000000 - 0x00001000	Master Boot Record (MBR)

nRF52 Series devices

The following figure displays the default memory layout of a 512 kB nRF52 device:



Memory layout

The following table shows the memory layout on a 512 kB nRF52 device with the S132 SoftDevice v2.0.0-7.alpha:

Memory range	Usage
0x0007A000 - 0x00080000	DFU Bootloader, data, and MBR parameters
0x0001B000 - 0x0007A000	Application code, free/swap, and application data
0x00001000 - 0x0001B000	SoftDevice
0x00000000 - 0x00001000	Master Boot Record (MBR)

4.2. nRF5 SDK v11.0.0-2.alpha

Relocating the bootloader

*This information applies to the **nRF51 Series** only.*

*This information applies to the following SoftDevices: **S130, S132***

The start address of the bootloader is stored in the UICR.BOOTLOADERADDR register, which is located at address 0x10001014. In the DFU bootloader example, writing the UICR.BOOTLOADERADDR is linked into the HEX file, so that it is automatically written when programming the bootloader (see Programming the DFU bootloader).

Note that UICR.BOOTLOADERADDR can only be written if you use `nrfjprog.exe` as the tool for flash programming. Always erase the device before programming the bootloader, so that the value can be set correctly.

The following code in `bootloader_util_arm.c` writes the bootloader start address to UICR.BOOTLOADERADDR:

```
uint32_t m_uicr_bootloader_start_address __attribute__((at(NRF_UICR_BOOT_START_ADDRESS))) =
        BOOTLOADER_REGION_START;
```

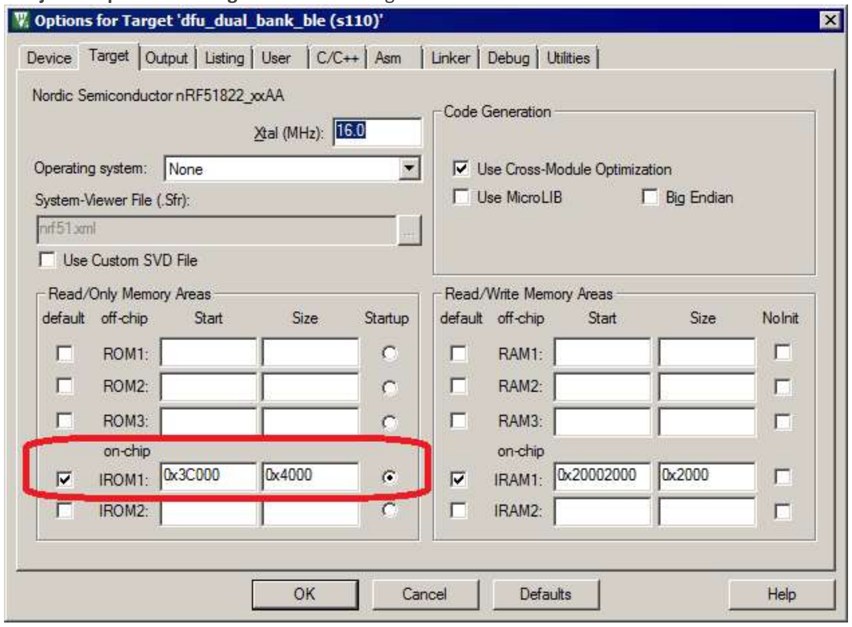
BOOTLOADER_REGION_START must point to the correct location of the bootloader in the flash. In this example, this location is 0x0003C000.

You can change the start address of the bootloader if the size of your bootloader image exceeds the available memory. For example, changing the start address from 0x0003C000 to 0x00038000 increases the available bootloader size to 0x8000 (instead of 0x4000).

When selecting a new start address, keep in mind that all start addresses must be page aligned. The page size is 0x400 (1024) bytes. Also, start address and size of the memory for the bootloader must add up to 0x00040000 bytes (the end of the application area in the flash).

Complete the following steps to relocate the bootloader to a new start address:

1. Erase the device.
2. Program the SoftDevice.
3. Change the value of BOOTLOADER_REGION_START in `dfu_types.h`.
4. Compile the bootloader.
5. In Keil, select **Project** > **Options for Target '*xxx*'** and change the IROM1 start address and size:



Nordic Semiconductor nRF51822_xxAA

Xtal (MHz): 16.0

Operating system: None

System-Viewer File (.Sfr): nrf51.xml

☐ Use Custom SVD File

Code Generation

☑ Use Cross-Module Optimization

☐ Use MicroLIB ☐ Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x3C000	0x4000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x20002000	0x2000	☐
☐	IRAM2:			☐

DFU bootloader start address setting in Keil

6. Program the bootloader.

4.3. nRF5 SDK v11.0.0-2.alpha

Dual-bank and single-bank updates

*This information applies to the following SoftDevices: **S130, S132***

To safely perform a Device Firmware Update, the new firmware image should not be copied to the final location in memory until it has been validated. This ensures that only complete and valid images are activated. If an error occurs during the transfer, the firmware should not be updated.

This process of storing the received firmware in one memory location (bank 1) and then copying it to the intended memory location (bank 0) later is called a *dual-bank update*.

Application images can alternatively be transferred in a *single-bank update*. In this process, the new application directly overwrites the existing application. If an error occurs during the transfer, both the old and the new application will be corrupt. However, a single-bank update makes it possible to transfer bigger applications that cannot be stored in addition to the original application.

All updates of the bootloader, the SoftDevice, or a combination of bootloader and SoftDevice are performed as dual-bank updates. Application updates can be performed as dual-bank or single-bank updates.

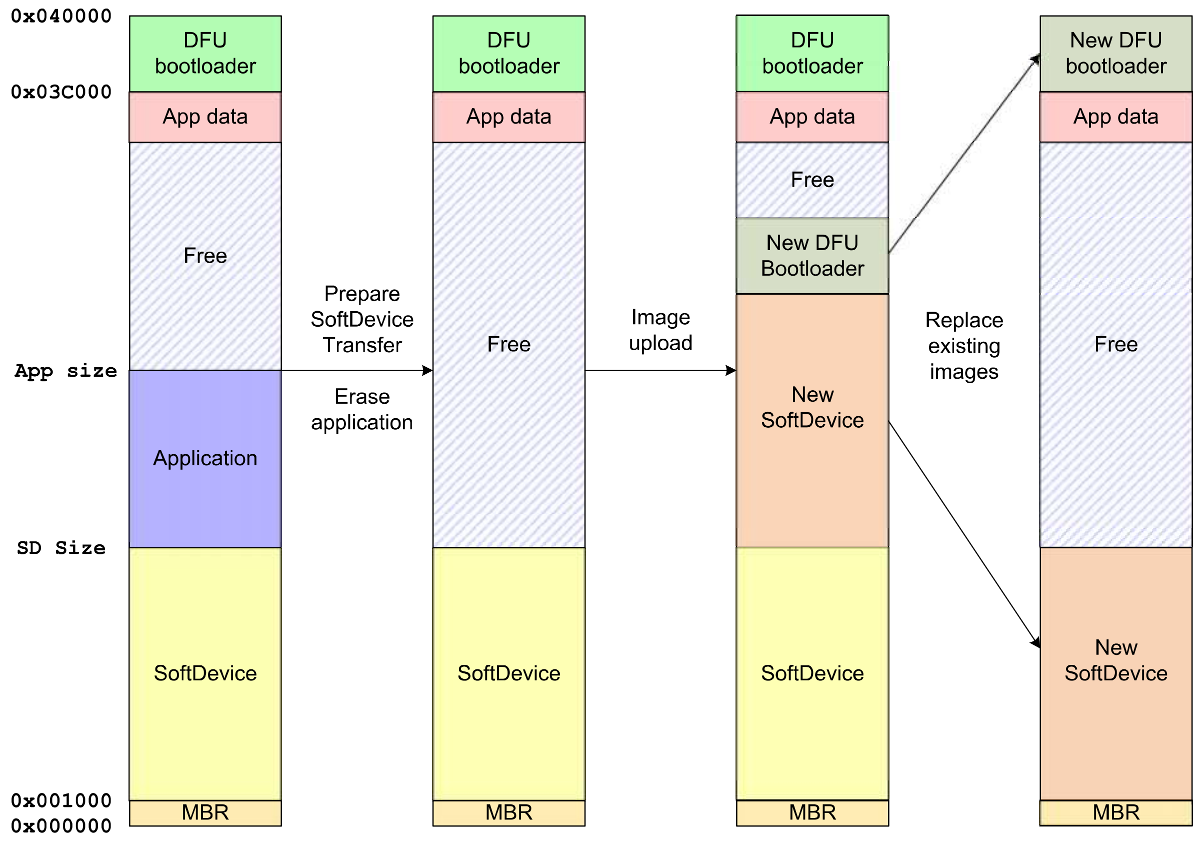
Dual-bank updates

During a dual-bank update, the existing bootloader, SoftDevice, or application is preserved until it is replaced by the new firmware image. The update process differs slightly depending on the type of image that is transferred.

SoftDevice (with or without bootloader)

When updating the SoftDevice or a combination of bootloader and SoftDevice, any application that is present on the device will be erased. The memory area between the end of the existing SoftDevice and the beginning of the existing bootloader is used to store the received image before it is copied to replace the existing SoftDevice and maybe bootloader.

The following figure shows the DFU process for a combined image of bootloader and SoftDevice. If the transferred image contains only a SoftDevice, the general process is the same, but only the SoftDevice is replaced.



DFU flash operations for SoftDevice and bootloader update

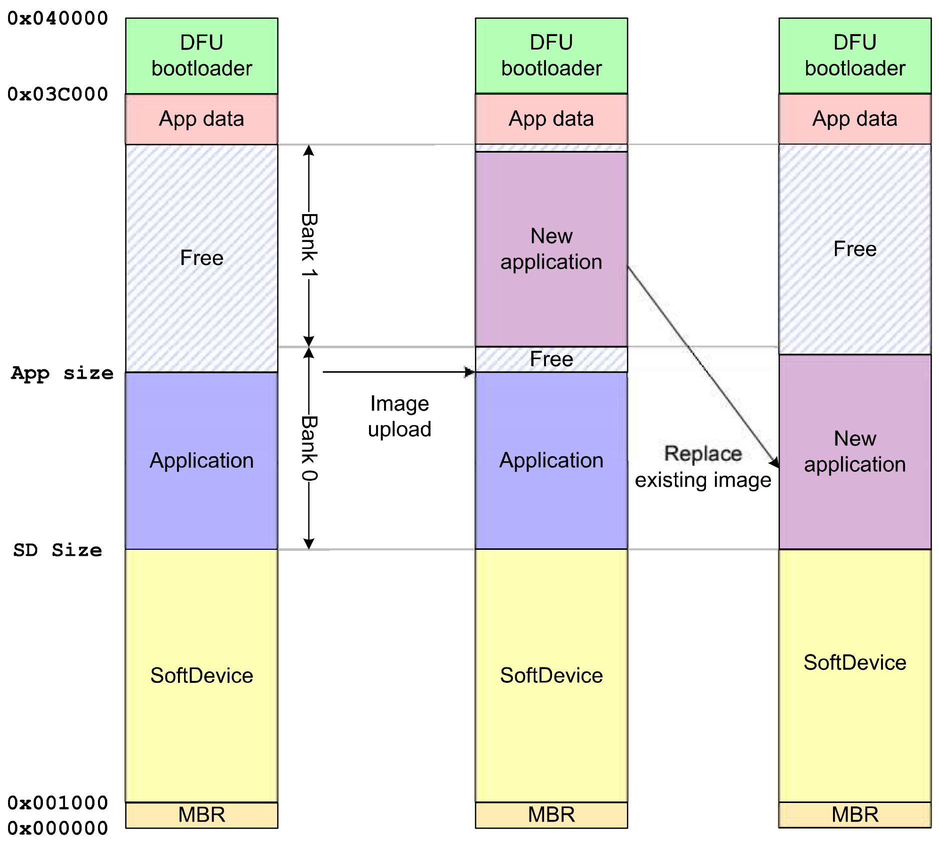
First, the existing application is erased to prepare for the DFU transfer. The transferred image is then stored between the current SoftDevice and bootloader. Existing application data can be retained; see Preserving application data for more information. Finally, the new SoftDevice and, if applicable, the new bootloader are validated and copied to replace the existing firmware.

Application or bootloader

When updating the application or the bootloader (without SoftDevice update), the existing application is retained during the update process.

The memory area between the end of the SoftDevice and the beginning of the bootloader (or the beginning of the application data, see Preserving application data) is divided into two banks. Bank 0 holds the existing application, and bank 1 is used to store the received image.

The following figure shows the DFU process for an application. A bootloader update works in the same way, except that the existing bootloader is replaced instead of the existing application.



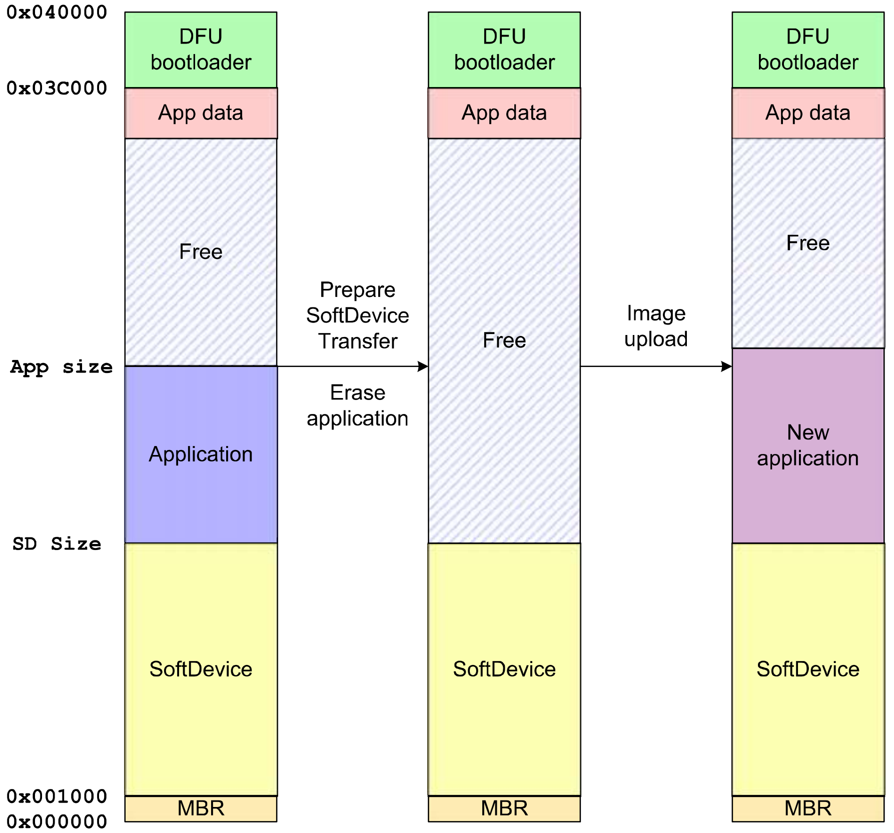
DFU flash operations for a dual-bank application update

The original application is located in memory bank 0. First, the bootloader erases bank 1. Existing application data can be retained; see Preserving application data for more information. The transferred image is then stored in bank 1. After all packets of the new application are received, both the old and the new application are present in the memory. This ensures that fallback to the old application is possible if the new application cannot be activated. If the new application can be activated, bank 0 is erased and the new application is copied from bank 1 to bank 0. Bank 1 is not erased until the bootloader initializes again.

Single-bank updates

In a single-bank update, the existing application is replaced with the new application during the transfer of the image. Single-bank updates are available only for application updates, because if an error occurs and the device is left without a valid application, you can recover it by uploading a valid application again. If the device is left without a valid bootloader or SoftDevice, you can recover it only by attaching a flash tool and updating the device with a flash programmer.

The following figure shows the DFU process for an application in single-bank mode.



DFU flash operations for a single-bank application update

First, the existing application is erased to prepare for the DFU transfer. The transferred image is then stored at the location of the old application, thus between the current SoftDevice and bootloader. Existing application data can be retained; see Preserving application data for more information. When the transfer is completed, the bootloader will validate the new application. If it is valid, the bootloader will start it. If it is not valid, the bootloader will reset, start in DFU mode, and wait for a new image to be uploaded.

4.4. nRF5 SDK v11.0.0-2.alpha

Preserving application data

*This information applies to the following SoftDevices: **S130, S132***

By default, all application data (for example bonding information, system attributes, or data that the application wants to preserve between resets) will be erased during a Device Firmware Update. However, you can configure the DFU bootloader to retain existing application data if your application can work with this data. In this case, new application data will be appended to the existing data.

To preserve application data during a DFU, configure a value for DFU_APP_DATA_RESERVED in `dfu_types.h`. The default value is 0x0000:

```
#define DFU_APP_DATA_RESERVED                0x0000
```

The value must be a multiple of the flash page size, for example 0x0400, 0x0800, 0x0C00, and so on. The default value of 0x0000 means that no application data is preserved. If you set the value to 0x1000, for example, 4096 bytes (4 pages) of application data will be reserved.

Application data is stored in the memory area between the application and the bootloader, right before the beginning of the bootloader.

Transport layers

*This information applies to the following SoftDevices: **S130, S132***

The DFU process is designed to be transport agnostic, as long as a reliable transport layer is used. The DFU bootloader example supports two transport layers: *Bluetooth* low energy (BLE) and UART (HCI).

Different types of messages are used to, for example, initialize a DFU, to send the init packet, and to send the actual data. Depending on the transport layer that is use, packets might be written to different channels. For example, BLE uses different end points for control packets (DFU messages) and data packets (init packet and firmware fragments), while serial transport uses different packet formats only.

All transports have different fixed maximum sizes for a data unit. To transfer data that exceeds these maximum sizes, the data must be split up into smaller packets. The DFU target will then reassemble the firmware fragments. However, the DFU process does not implement sequence numbers or a DFU packet loss mechanism, so the transport layer must submit the DFU packets in the right order and one at a time. The DFU example does not detect out-of-sequence packets, and the transport must ensure that all packets are delivered in the right order.

For more information about the BLE and serial transport, see the following page:

- BLE DFU Profile
- BLE DFU Service
- Serial (HCI) packet format

BLE DFU Profile

*This information applies to the following SoftDevices: **S130, S132***

The Device Firmware Update (DFU) Profile is an example for a proprietary profile that can be used to update firmware files using *Bluetooth* low energy. This profile is not a standard profile. It is defined by Nordic Semiconductor to demonstrate how a typical Device Firmware Update can be achieved.

The DFU Profile is used to transfer application, SoftDevice, or bootloader images from a BLE central (for example, a computer or a smartphone) to a peripheral (for example, a Heart Rate Sensor) that supports Device Firmware Updates using the DFU Service.

Profile dependencies: This profile requires the Generic Attribute Profile (GATT).

Configuration

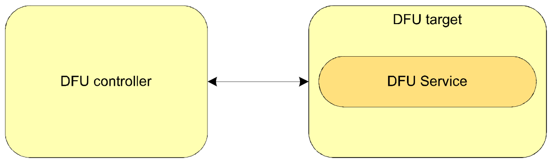
Roles

The profile defines two roles: DFU target and client (also referred to as DFU controller).

- The DFU target implements a GATT Server. This is the device that is updated by the DFU controller.
- The DFU controller implements a GATT client. This is the device that uploads a new firmware image to the DFU target.

Role/service relationships

The following diagram shows the relationships between services and the two profile roles.



Relationship between profile roles and services

A DFU target instantiates one instance of the DFU Service.

DFU controller role requirements

The DFU controller shall support the BLE DFU Service.

Service	DFU controller
Device Firmware Update Service	M

General error handling

The DFU controller shall be tolerant when receiving the following ATT Error Code defined in CSS Part B, Section 1.2 of Supplement to the *Bluetooth* Core Specification, Version 3 or later:

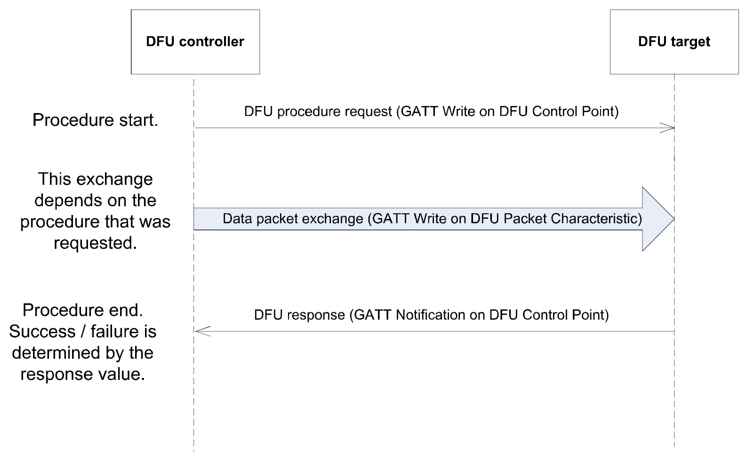
- Client Characteristic Configuration Descriptor Improperly Configured

DFU image transfer procedure

The DFU controller requests one of the procedures that are defined for a Device Firmware Update using the *DFU Control Point*. Each procedure requested by the controller is characterized by a request on the DFU Control Point, using the GATT Write procedure. This procedure may be followed by additional writes on the *DFU Packet* characteristic. The request to activate the image and reset the device ends the procedure. The DFU target for the procedure responds to this request using GATT notifications on the control point. The response code in the response packet contains information about the success or failure of the procedure. In case of failure, the response code provides an indication on the possible reason for the failure. See DFU Control Point for more details.

The DFU target processes one request at a time. Therefore, if a request is received on the target while it is already processing another request, an ATT error response ("Procedure Already In Progress") is received on the DFU controller. Asynchronous status information and information about the size of the received image is sent by the DFU target as GATT notifications on the control point.

The following message sequence chart shows a typical DFU Control Point procedure:


DFU controller
DFU target
Procedure start.
DFU procedure request (GATT Write on DFU Control Point)
This exchange depends on the procedure that was requested.
Data packet exchange (GATT Write on DFU Packet Characteristic)
Procedure end. Success / failure is determined by the response value.
DFU response (GATT Notification on DFU Control Point)

Typical DFU Control Point procedure

All defined DFU procedures are listed below:

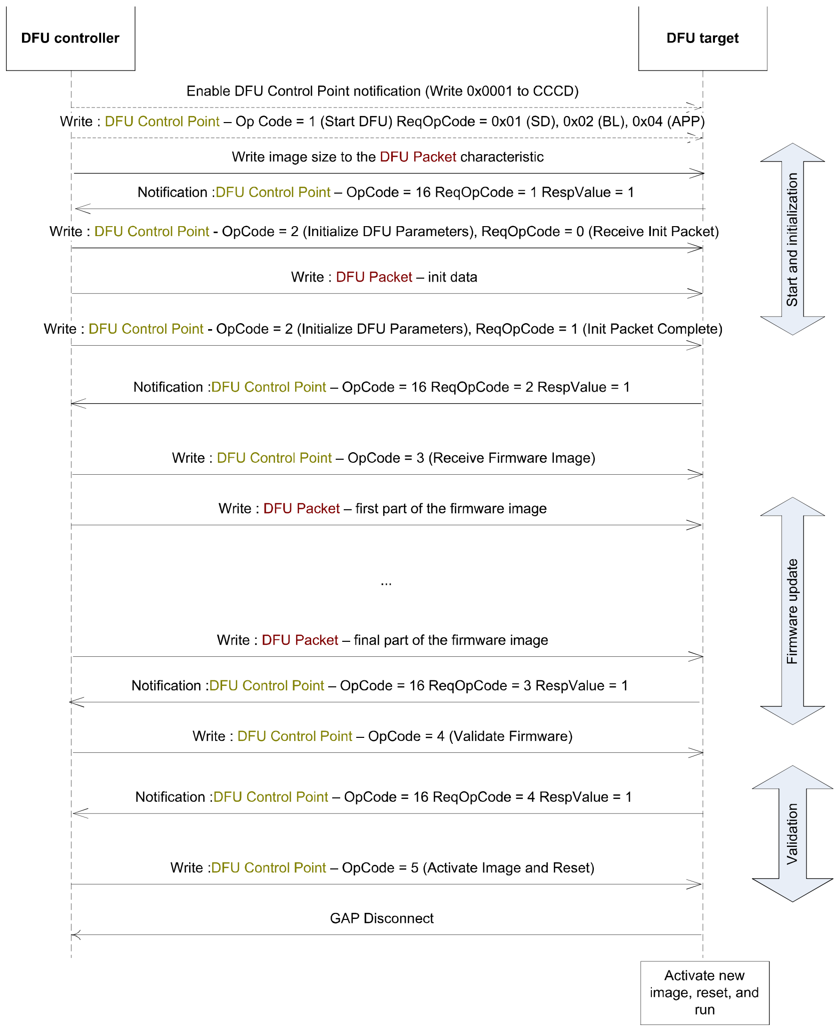
DFU procedure	Description
Start DFU	Procedure to prepare the device for uploading the firmware. The size of the firmware image is provided in the request parameter. The response of the DFU target indicates if the device is ready for the next stage of the firmware update. The Start DFU procedure must be followed by a byte indicating the type of update.
Receive Init Data	Procedure to exchange information that must be exchanged before uploading the new firmware. Such information includes the device type, the revision type, the application version, hashs/public keys, and so on. See Safety-checking the image for more information.
Receive Image Data	Procedure to upload firmware, fragmented into DFU packets. The maximum size of each packet is (ATT_MTU - 3) octets. DFU packets can be of variable length with a minimum size of 1 octet; the length must be a multiple of the word size. The firmware is expected in binary format.
Validate	Procedure to validate the updated firmware image. The current implementation supports only CRC validation in addition to size validation (which is always performed). CRC validation is optional and uses the information provided in the Receive Init Data procedure.
Activate Image & Reset	Procedure to activate the new firmware and restart the device with the new firmware. This procedure results in a GAP Disconnect.
System Reset	

	Procedure to reset the system. This procedure result in a GAP Disconnect. This procedure can be requested at any stage. The system will then restart with the old application if the device had an existing application. Otherwise, the system will restart in DFU mode.
Report Received Image Size	Procedure to request the size of the received image. This procedure is particularly useful on reconnection after a link loss. See Link loss procedure for more information.
Packet Receipt Notification	Procedure to request notifications every time after receiving a certain number of packets. The number of packets is specified in the request by the controller. This procedure is not mandatory.

See DFU Control Point for more information about the procedures.

When the DFU target is in DFU mode, it is in *GAP General Discoverable mode* and is *connectable*. The 128-bit DFU Service UUID is advertised in the advertisement data, along with the full name (DfuTarg). When the DFU target is connected to the DFU controller, the DFU process can be started by requesting a Start DFU procedure.

The following message sequence chart shows a typical firmware update exchange between a DFU controller and a DFU target:


DFU controller
DFU target
Enable DFU Control Point notification (Write 0x0001 to CCCD)
Write : DFU Control Point – Op Code = 1 (Start DFU) ReqOpCode = 0x01 (SD), 0x02 (BL), 0x04 (APP)
Write image size to the DFU Packet characteristic
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 1 RespValue = 1
Write : DFU Control Point - OpCode = 2 (Initialize DFU Parameters), ReqOpCode = 0 (Receive Init Packet)
Write : DFU Packet – init data
Write : DFU Control Point - OpCode = 2 (Initialize DFU Parameters), ReqOpCode = 1 (Init Packet Complete)
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 2 RespValue = 1
Write : DFU Control Point – OpCode = 3 (Receive Firmware Image)
Write : DFU Packet – first part of the firmware image
...
Write : DFU Packet – final part of the firmware image
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 3 RespValue = 1
Write : DFU Control Point – OpCode = 4 (Validate Firmware)
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 4 RespValue = 1
Write :DFU Control Point – OpCode = 5 (Activate Image and Reset)
GAP Disconnect
Activate new image, reset, and run
Start and initialization
Firmware update
Validation

Transfer of an image to the DFU target

Idle mode procedure

If no connection is established with the DFU target after 60 seconds, the target device will restart in normal mode with the old application if the device had an existing application. Otherwise, the system will restart in DFU mode.

Image validation procedure

The post-validate function dfu_init_postvalidate in `dfu_init_template` validates the updated firmware image.

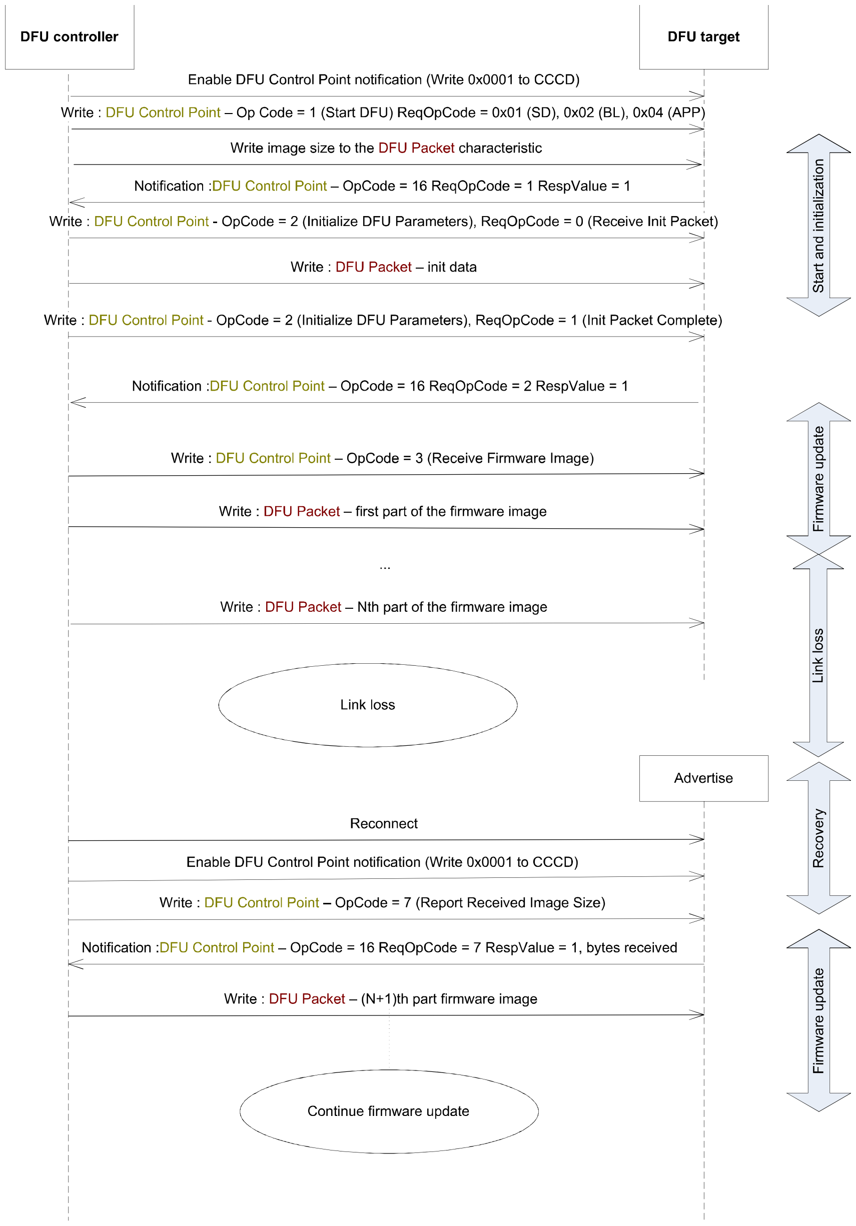
Idle connection procedure

When the bootloader example application detects that the connection has been idle for a certain period, thus if the DFU controller has not written any packets to continue the firmware update, the update is stopped. The connection is terminated and the previously valid application is started. If there is no valid application, the device remains in bootloader mode and waits for a reconnection from the DFU controller.

Link loss procedure

When the bootloader example application detects a link loss, the DFU state and image size is stored. When the controller connects again, the transfer can resume from where it stopped. The controller can request information about the size of the received image using the Report Received Image Size procedure and start thetransfer by applying the necessary offset. In the disconnected state after a link loss, the Idle mode procedure applies.

The following message sequence chart shows a typical link loss recovery procedure:


DFU controller
DFU target
Enable DFU Control Point notification (Write 0x0001 to CCCD)
Write : DFU Control Point – Op Code = 1 (Start DFU) ReqOpCode = 0x01 (SD), 0x02 (BL), 0x04 (APP)
Write image size to the DFU Packet characteristic
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 1 RespValue = 1
Write : DFU Control Point - OpCode = 2 (Initialize DFU Parameters), ReqOpCode = 0 (Receive Init Packet)
Write : DFU Packet – init data
Write : DFU Control Point - OpCode = 2 (Initialize DFU Parameters), ReqOpCode = 1 (Init Packet Complete)
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 2 RespValue = 1
Write : DFU Control Point – OpCode = 3 (Receive Firmware Image)
Write : DFU Packet – first part of the firmware image
...
Write : DFU Packet – Nth part of the firmware image
Link loss
Advertise
Reconnect
Enable DFU Control Point notification (Write 0x0001 to CCCD)
Write : DFU Control Point – OpCode = 7 (Report Received Image Size)
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 7 RespValue = 1, bytes received
Write : DFU Packet – (N+1)th part firmware image
Continue firmware update
Start and initialization
Firmware update
Link loss
Recovery
Firmware update

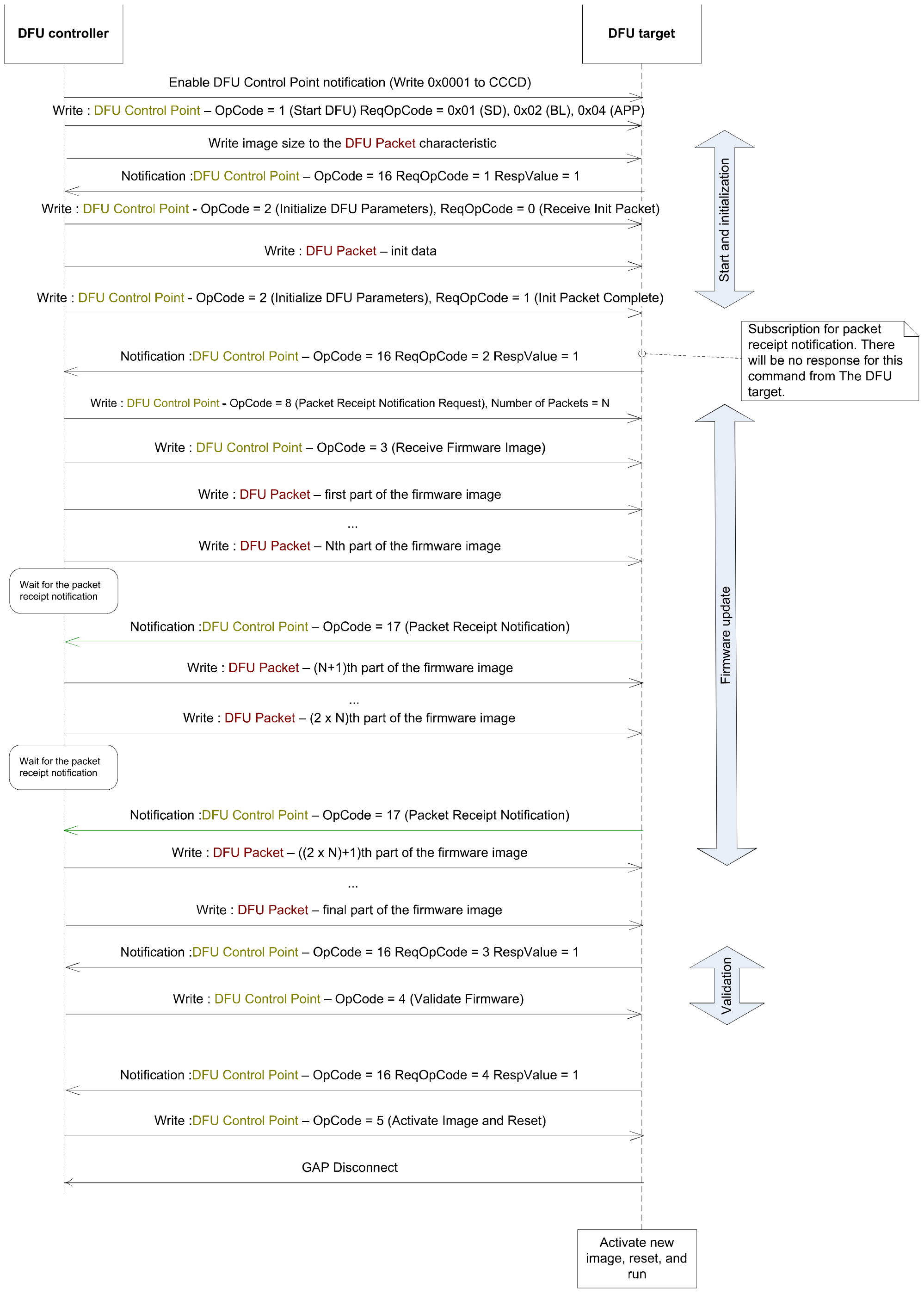
Recovery after link loss

Packet Receipt Notification procedure

The DFU controller can subscribe to notifications from the DFU target, to be notified every time a given number of firmware packets has been received. To enable such notifications, the DFU controller must write the Op Code 0x08 (Packet Receipt Notification Request) followed by the number of packets to the DFU Control Point. The DFU target will then send a notification of the DFU Control Point with Op Code 0x11 (Packet Receipt Notification) every time the specified number of firmware packets has been received. The DFU controller should wait for this notification every time after sending the specified number of firmware packets.

As part of the packet receipt notification, the DFU target also sends the total number of bytes of firmware data received so far. This information can be used for consistency checks by the DFU controller.

The following message sequence chart shows a typical Packet Receipt Notification procedure:


DFU controller
DFU target
Enable DFU Control Point notification (Write 0x0001 to CCCD)
Write : DFU Control Point – OpCode = 1 (Start DFU) ReqOpCode = 0x01 (SD), 0x02 (BL), 0x04 (APP)
Write image size to the DFU Packet characteristic
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 1 RespValue = 1
Write : DFU Control Point - OpCode = 2 (Initialize DFU Parameters), ReqOpCode = 0 (Receive Init Packet)
Write : DFU Packet – init data
Write : DFU Control Point - OpCode = 2 (Initialize DFU Parameters), ReqOpCode = 1 (Init Packet Complete)
Start and initialization
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 2 RespValue = 1
Subscription for packet receipt notification. There will be no response for this command from The DFU target.
Write : DFU Control Point - OpCode = 8 (Packet Receipt Notification Request), Number of Packets = N
Write : DFU Control Point – OpCode = 3 (Receive Firmware Image)
Write : DFU Packet – first part of the firmware image
...
Write : DFU Packet – Nth part of the firmware image
Wait for the packet receipt notification
Notification :DFU Control Point – OpCode = 17 (Packet Receipt Notification)
Write : DFU Packet – (N+1)th part of the firmware image
...
Write : DFU Packet – (2 x N)th part of the firmware image
Wait for the packet receipt notification
Notification :DFU Control Point – OpCode = 17 (Packet Receipt Notification)
Write : DFU Packet – ((2 x N)+1)th part of the firmware image
...
Write : DFU Packet – final part of the firmware image
Firmware update
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 3 RespValue = 1
Write : DFU Control Point – OpCode = 4 (Validate Firmware)
Validation
Notification :DFU Control Point – OpCode = 16 ReqOpCode = 4 RespValue = 1
Write :DFU Control Point – OpCode = 5 (Activate Image and Reset)
GAP Disconnect
Activate new image, reset, and run

Packet Receipt Notification procedure

Security considerations

All supported characteristics specified by the DFU Service on the target are set to Security Mode 1 and Security Level 1.

This document was last updated on Fri Dec 18 2015.
Please send us your feedback about the documentation! For technical questions, visit the Nordic Developer Zone.

5.2. nRF5 SDK v11.0.0-2.alpha

BLE DFU Service

*This information applies to the following SoftDevices: **S130, S132***

The Device Firmware Update (DFU) Service exposes necessary information to perform Device Firmware Updates on the device. This service is not a service defined by the *Bluetooth* SIG, but a proprietary service defined by Nordic Semiconductor to demonstrate a typical Device Firmware Update on an nRF5 device.

The DFU Service does not depend on any other services. Support for the following GATT sub-procedures is mandatory for this service:

- Write Characteristic Value
- Notifications
- Read Characteristic Descriptors
- Write Characteristic Descriptors

The DFU GATT Service can operate only on *Bluetooth* low energy as transport.

The DFU Service does not define any new error codes for the Attribute Protocol. Data exchange is in little endian (LSB first) order.

This service is instantiated as a primary service in DFU mode.

Proprietary service UUID

The assigned service UUID is 0x1531 over proprietary base. See the following table for Nordic Semiconductor's UUID:

Description	Number base
Company identifier:	0x0059
UUID base:	0x23, 0xD1, 0xBC, 0xEA, 0x5F, 0x78, 0x23, 0x15, 0xDE, 0xEF, 0x12, 0x12, 0x00, 0x00, 0x00, 0x00
Service UUID start:	0x1530
Characteristic UUID start:	0x1531

Service characteristics

The DFU Service exposes one instance of the characteristics listed in the following table. The service does not impose any security requirements.

Characteristic name	Requirement	Mandatory properties	Description
DFU Packet	M	WriteWithoutResponse	See DFU Packet.
DFU Control Point	M	Write, Notify	See DFU Control Point.

DFU Packet

The UUID of the DFU Packet characteristic is 0x1532 over proprietary base.

This characteristic receives data for Device Firmware Updates as DFU packets. DFU packets can contain different types of data, depending to the Op Code that is written to the DFU Control Point before the DFU packet is transmitted.

The size of each packet must be between 1 and (ATT_MTU - 3) octets. Packets must be in little endian (LSB first) order.

The following table defines the format for a DFU packet:

Names	Field requirement	Format	Additional information
DFU Packet	Mandatory	uint8	This field may be repeated up to a maximum of 20 times, which means that the maximum length of this characteristic is 20 bytes.

(Minimum and maximum values are not applicable.)

Image size

After writing "Start DFU" (0x01) to the DFU Control Point, you must write the image size to the DFU Packet characteristic.

The image size must be written in the following format:

```
<Length of SoftDevice><Length of bootloader><Length of application>
```

All lengths must be uint32. If a length is not present (for example, if only the SoftDevice is updated), the length should be given as 0. For example:

```
<Length of SoftDevice> 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

Init packet

After writing "Initialize DFU Parameters" (0x02) to the DFU Control Point, you must write an init packet to the DFU Packet characteristic.

See Safety-checking the image for information about the format of the init packet.

Image data

After writing "Receive Firmware Image" (0x03) to the DFU Control Point, you must write packets containing image data to the DFU Packet characteristic.

The firmware image can be split up in multiple DFU packets. The full image is transferred by writing each fragment as DFU packet to the DFU Packet characteristic.

DFU Control Point

The UUID of the DFU Control Point characteristic is 0x1531 over proprietary base.

The DFU Control Point characteristic is used to control the state of the Device Firmware Update process. All DFU procedures are requested by writing to this characteristic. A response that marks the end of the procedure is received as a notification.

The following table shows control point procedure operation codes and the respective parameters:

Names	Field requirement	Format	Additional information
Op Code	Mandatory	uint8	Enumerations:

Key	Value	Requirement	Description
0	Reserved for future use		
1	Start DFU	C4	Initiate the firmware update procedure. The response to this Op Code is a notification of the control point with Op Code 0x10, followed by the request Op Code 0x01 and the appropriate Response Value.
2	Initialize DFU Parameters	C5	Prepare to receive init packets. The response to this Op Code is a notification of the control point with Op Code 0x10, followed by the request Op Code 0x02 and the appropriate Response Value.
3	Receive Firmware Image		Prepare to receive the firmware data. The response to this Op Code is a notification of the control point with Op Code 0x10, followed by the request Op Code 0x03 and the appropriate Response Value.
4	Validate Firmware		Validate the received firmware. The response to this Op Code is a notification of the control point with Op Code 0x10, followed by the appropriate Response Value.
5	Activate Image and Reset		Activate the previously received image and perform a system reset. There is no response to this Op Code.
6	Reset System		Perform a system reset. There is no response to this Op Code.
7	Report Received Image Size		Request the DFU target to report the total number of bytes of firmware data received (excluding the start data and init data). The response to this Op Code is a notification of the control point with Op Code 0x10, followed by the request Op Code 0x07, the appropriate Response Value, and the number of bytes in the Response Parameter.
8	Packet Receipt Notification Request	C1	Request the DFU target to enable/disable notifications of the control point characteristic each time the specified number of packets containing firmware data has been

			received. There is no response to this Op Code.
16	Response Code	C2	The Response Code is followed by the Request Op Code, the Response Value, and optionally the Response Parameter.
17	Packet Receipt Notification	C3	A notification sent by the DFU target indicating that a new set of the preconfigured number of firmware data packets has been received.
9-15	Reserved for future use		
18-255	Reserved for future use		

Number of Packets **Information:** Parameter value for the Packet Receipt Notification Request Op Code	C1	uint16	Number of packets of firmware data to be received by the DFU target before sending a new Packet Receipt Notification (control point notification with Op Code = 7). If this value is 0, the packet receipt notification will be disabled by the DFU target.
Request Op Code **Information:** Parameter value for the Response Code Op Code	C2	uint8	Refer to the Op Code table above for additional information on the possible values for this field.
Response Value **Information:** C2: This field is mandatory for the Response Code Op Code. Otherwise, this field is excluded.	C2	uint8	Enumerations:

Key	Value	Description
0	Reserved for future use	
1	Success	Response for successful operation.
2	Invalid State	The DFU controller has performed an operation that is not valid in the current state of the firmware update process.
3	Not Supported	The previous operation performed by the DFU controller or the data sent by the DFU controller is not supported.
4	Data Size Exceeds Limit	The DFU controller is trying to send more firmware data than expected.
5	CRC Error	A CRC error has occurred. This Response Value is used only by the Validate Firmware procedure.
6	Operation Failed	Response if the requested procedure failed.
7-255	Reserved for future use	

Response Parameter **Information:** C2: This field is optional for the Response Code Op Code. Otherwise, this field is excluded.	C2	variable	Note: The Response Parameter of the response to the control point is a variable length field to allow a list of different values defined by the service specification.
Number of Bytes of Firmware Image Received **Information:** C3: This field is present if the Op Code is 0x11 (Packet Receipt Notification).	C3	uint32	Number of bytes of firmware data (excluding the start and init data) received by the DFU target at the given point of time.
DFU Image Type **Information:** C4: This field is present if the Op Code is 0x01 (Start DFU). This field is parsed as bit field where each bit that is set indicates the image transferred for the requested DFU.	C4	uint8	Enumerations:

Key	Value	Description
0x00	No Image	No image will be updated.
0x01	SoftDevice	A SoftDevice image will be transferred.
0x02	Bootloader	A bootloader image will be transferred.
0x03	SoftDevice Bootloader	A SoftDevice with Bootloader image will be transferred.
0x04	Application	An application image will be transferred.
0x05-0x07	Other image combinations	Currently not supported.
0x08-0xFF	Reserved for future use	

DFU Init Packet **Information:** C5: This field is present if the Op Code is 0x02 (Initialize DFU Parameters).	C5	uint8	Enumerations: Key / Value / Description 0x00 / Receive Init Packet / Ready to receive the DFU init packet. 0x01 / Init Packet Complete / Transmission of the DFU init packet is complete.

(Minimum and maximum values are not applicable.)

General error handling procedures

If an Op Code is written to the DFU Control Point characteristic and the Client Characteristic Configuration Descriptors of either or both of the DFU Control Point or the DFU Status Report are not configured for notifications, the DFU target will return an error response. The Attribute Protocol Application error code of this response will be set to "Client Characteristic Configuration Descriptor Improperly Configured" (as defined in CSS Part B, Section 1.2 of Supplement to the *Bluetooth* Core Specification, Version 3 or later).

5.3. nRF5 SDK v11.0.0-2.alpha

Serial (HCI) packet format

*This information applies to the following SoftDevices: **S130, S132***

When you use serial (HCI) transport to send Device Firmware Updates, there are four different types of packets that you must send to the DFU bootloader in sequence:

1. One start packet to initiate the firmware update procedure.
2. One init packets that is used to safety-check the new image.
3. Any number of data packets that contain the new image.
4. One stop packet to notify the bootloader that the transfer has completed and the image should be activated.

Start packet

The first packet that the DFU bootloader must receive is the start packet. This packet informs the DFU bootloader about the kind of update and the length of the binary image. The length of the image is sent as an unsigned 32-bit integer in LSB first order.

The following table gives an example of a DFU start packet. Values in italics are example values and will be different for your update.

Value	Description
0xC0	SLIP start/stop
0xD1 0x4E 0x01 0xE0	Packet header
0x03 0x00 0x00 0x00	DFU bootloader start packet
0x04 0x00 0x00 0x00	Type of the image to be transferred (*0x04* for an application image)
0x00 0x00 0x00 0x00	Length of the new SoftDevice image, or 0 if no SoftDevice image is transferred
0x00 0x00 0x00 0x00	Length of the new bootloader image, or 0 if no bootloader image is transferred
0x0C 0x5D 0x00 0x00	Length of the new application image, or 0 if no application image is transferred
0xC6 0xFC	CRC check
0xC0	SLIP start/stop

The following values can be used to define the type of the image:

Value	Image type
0x01	SoftDevice
0x02	Bootloader
0x03	Combined SoftDevice and bootloader
0x04	Application

Init packet

The init packet contains information about the application that is transferred. The DFU bootloader checks this information to determine if the image is valid for the device. If it is, it prepares (erases) the flash memory to accommodate the new image.

The following table gives an example of a DFU init packet. Values in italics are example values and will be different for your update.

Value	Description
0xC0	SLIP start/stop
0xDA 0x4E 0x01 0xD7	Packet header
0x01 0x00 0x00 0x00	DFU bootloader init packet
0xFF 0xFF	Device type
0xFF 0xFF	Device revision
0xFF 0xFF 0xFF 0xFF	Application version
0x01 0x00	Length of the list of valid SoftDevices
0xFE 0xFF	List of valid SoftDevices
0xCA 0x2C	CRC of the image that will be transferred
0x00 0x00	Filler
0x7E 0xBD	CRC check
0xC0	SLIP start/stop

Data packet

Data packets contain the actual image that is transferred. The maximum size of a data packet is 512 bytes + header.

The following table gives an example of a DFU data packet. Values in italics are example values and will be different for your update.

Value	Description
0xC0	SLIP start/stop
0xE3 0x4E 0x20 0xAF	Packet header
0x04 0x00 0x00 0x00	DFU bootloader data packet
0xA8 0x2F 0x00 0x20 …	Image data, max. 512 bytes
0x51 0xC7	CRC check
0xC0	SLIP start/stop

Stop packet

When the new image has been transferred to the nRF5 IC, the image must be copied and activated. The stop packet informs the DFU bootloader that the image has been transferred completely and can now be activated.

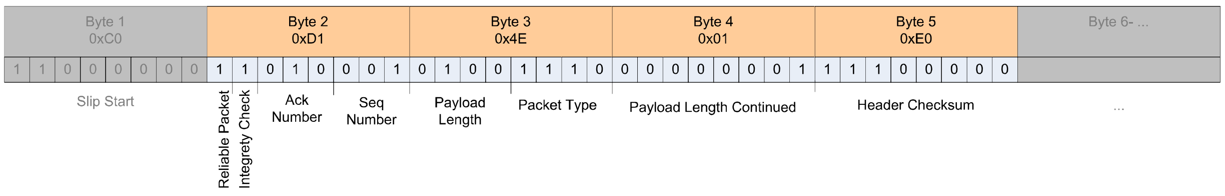
The following table gives an example of a DFU data packet. Values in italics are example values and will be different for your update.

Value	Description
0xC0	SLIP start/stop
0xC8 0x4E 0x00 0xEA	Packet header
0x05 0x00 0x00 0x00	DFU bootloader stop packet
0x36 0x0A	CRC check
0xC0	SLIP start/stop

Packet header

See the *Bluetooth* Core Specification (Volume 4, Part D, Chapter 4) for detailed information about the packet header.

The following figure illustrates the packet header for the example Start packet:



Packet header