nRF Util

v6.1.0

User Guide

4399_007 / 2021-11-17

NORDIC® SEMICONDUCTOR

Contents

Revision history

Date	Description
2021-11-17	• Added nRF52820 and nRF52805 to Table 2: `--family` settings on page 16 • Editorial changes
July 2020	Updated for nRF Util v6.1: • Added Generating HEX files for Zigbee on page 18
December 2019	Updated for nRF Util v6.0: • Updated tool name • Updated Python version • Updated Generating DFU packages on page 8 • Updated Performing a DFU on page 11 • Added DFU package combinations on page 8
March 2019	Updated for nrfutil v5.1.0: • Added DFU over ANT on page 11
February 2019	Updated for nrfutil v5.0.0: • Added Updating external applications on page 13
October 2018	Updated for nrfutil v4.0.0: • Added DFU over Zigbee on page 12
December 2017	Updated for nrfutil v3.4.0: • Added DFU over Thread on page 12 • Added DFU over a serial USB connection on page 14 • Updated the examples for Generating DFU packages on page 8 to show how to create unsigned packages
March 2017	Updated for nrfutil v2.2.0: • Added support for serial DFU (DFU over a serial UART connection on page 13) • Changed the location of the `dfu-cc.proto` file in the DFU bootloader implementation (Customizing the init packet on page 20)
November 2016	Updated for nrfutil v2.0.0
September 2016	First release

Previous versions

PDF files for relevant previous versions are available here:

- nrfutil User Guide v1.0 (corresponds to nrfutil v1.5.0)

1 Introduction

The nRF Util application is a Python package and command-line utility that supports *Device Firmware Update (DFU)* and cryptographic functionality.

The nRF Util application and its library has the following features:

- *DFU* package generation
- Cryptographic key generation, management, and storage
- Bootloader settings generation
- *DFU* procedure on the following protocols:
  - *Bluetooth*® Low Energy
  - Serial over UART
  - Serial over USB
  - Thread unicast
  - Thread multicast
  - Zigbee
  - ANT™

There are two different *DFU* package formats:

- Legacy – Uses a simple structure and no security.
- Modern – Uses Google's protocol buffers for serialization and can be cryptographically signed.

The DFU package format transitioned from legacy to modern in nRF5 SDK v12.0.0. Depending on the SDK version that you are using, select a compatible release of this tool:

- Version 0.5.x generates legacy firmware packages compatible with nRF5 SDK v11.0.0 and earlier.
- Version 1.5.0 and later generate modern firmware packages compatible with nRF5 SDK v12.0.0 and later.
- Version 2.2.0 or later is required to generate a bootloader settings page that is compatible with nRF52840.
- Version 4.0.0 and later generate modern firmware packages compatible with nRF5 SDK v15.1.0 and later.
- Version 5.0.0 and later generate modern firmware packages compatible with nRF5 SDK v15.3.0 and later.

> **Note:** To create firmware images compatible with nRF SDK 12.0 to nRF SDK 15.0, use the `--no-backup` command when generating *DFU* settings.

See the DFU bootloader and BLE Secure DFU Bootloader example in the SDK documentation for more information about *DFU*s.

2 Installing nRF Util

You can install nRF Util from the Python Package Index (PyPI) or you can run or install it from the sources.

In both cases, the following prerequisites must be installed:

- Python 3.7 or later
- pip (see Installing Python Modules)

2.1 Installing from PyPI

nRF Util is available as a package in the Python Package Index (PyPI) and can be downloaded and installed directly using the Python installer program **`pip`**.

Enter the following command to install the latest published version from PyPI:

```
pip install nrfutil
```

This command installs nRF Util and all required packages.

When installing on macOS, you might get an error about the Python module **`six`**. In this case, enter the following command instead:

```
pip install --ignore-installed six nrfutil
```

If you are running nRF Util on Windows, the runtime libraries targeted during the library build must be present when running code using the library. The following errors indicate that the runtime libraries are not available:

- Missing `MSVC*120.DLL` or `MSVC*140.DLL`
- `RuntimeError: Could not load shared library <path>/pc_ble_driver_shared.dll : '[Error 193] %1 is not a valid Win32 application`

In this case, install the Visual C++ redistributable packages for Visual Studio 2013 or Visual Studio 2015. Select the version that corresponds to the architecture of your Python installation (x86 or x64).

2.2 Installing from sources

Download the sources from GitHub to install nRF Util.

In addition to Python and **`pip`**, installing nRF Util from the sources requires the Python setuptools. To upgrade to the latest version, run the following command:

```
pip install -U setuptools
```

If you want to create an executable for nRF Util, install pyinstaller:

```
pip install pyinstaller
```

Complete the following steps to install nRF Util from the sources.

1. Clone the nRF Util GitHub repository.
2. Open a command prompt in the folder where you cloned the repository and run `pip install -r requirements.txt` to install all prerequisites.

3. Set up nRF Util in one of the following ways:

   - Run nRF Util from the sources without installation:

     ```
     python nordicsemi/__main__.py
     ```

     The remainder of this document assumes that you have installed the tool and can call it with `nrfutil`. If you choose to run it without installation, always replace the `nrfutil` command with `python nordicsemi/__main__.py` and add the required command-line options.
   - Install the library to the local Python site-packages and script folder:

     ```
     python setup.py install
     ```
   - Generate a self-contained executable version of the utility:

     ```
     pyinstaller nrfutil.spec
     ```

     Note: Some anti-virus programs will stop **`pyinstaller`** from executing correctly when it modifies the executable file. In this case, configure your anti-virus program to ignore **`pyinstaller`**.

If you are running nRF Util on Windows, the runtime libraries targeted during the library build must be present when running code using the library. The following errors indicate that the runtime libraries are not available:

- Missing `MSVC*120.DLL` or `MSVC*140.DLL`
- `RuntimeError: Could not load shared library <path>/pc_ble_driver_shared.dll : '[Error 193] %1 is not a valid Win32 application`

In this case, install the Visual C++ redistributable packages for Visual Studio 2013 or Visual Studio 2015. Select the version that corresponds to the architecture of your Python installation (x86 or x64).

3 Displaying help

Add `--help` to any nRF Util command to display help about the command.

Help is context-sensitive. Enter `nrfutil --help` to get information about the general usage of nRF Util, or `nrfutil command --help` to display help for a specific *command*, which can also correspond to a protocol name.

For example, enter the following command to display help on the *DFU* over Bluetooth LE procedure:

```
nrfutil dfu ble --help
```

To display all available options for Zigbee, enter the following command:

```
nrfutil zigbee --help
```

4 Generating DFU packages

The **pkg** command generates a package to use for a *DFU*. The package contains the new firmware image, an init packet, and a manifest file that indicates the package format. The command can also be used to display the package contents.

Run **nrfutil pkg generate** to generate a zip file that you can use later with a mobile application or another tool to update the firmware of an nRF5 device. You can see available options by entering the following command:

```
nrfutil pkg generate --help
```

Run **nrfutil pkg display** to display the contents of a package.

For example, enter the following command to generate an unsigned package called `app_dfu_package.zip` from the application file `app.hex`:

```
nrfutil pkg generate --application app.hex app_dfu_package.zip
```

Enter the following command to generate a package called `app_dfu_package.zip` from the application file `app.hex` with application version 4 that requires hardware version 51 and SoftDevice S130 v2.0.0 (0x80) and is signed with the private key that is stored in `key.pem`:

```
nrfutil pkg generate --hw-version 51 --sd-req 0x80 --application-version 4 --application
 app.hex --key-file key.pem app_dfu_package.zip
```

Enter the following command to generate an unsigned debug package without version information from the application file `app.hex`:

```
nrfutil pkg generate --debug-mode --application app.hex app_dfu_package.zip
```

Enter the following command to display the contents of the created package:

```
nrfutil pkg display app_dfu_package.zip
```

The `--hw-version` option must correspond to the nRF5 device used.

The `--sd-req` option must correspond to the firmware ID of the SoftDevice present on the target device. Refer to the list of SoftDevice firmware IDs (under --sd-req) displayed by the `nrfutil pkg generate --help` command.

> **Note:** While Thread and Zigbee stacks do not use a SoftDevice, the `--sd-req` option is required for compatibility. Any value provided for this option is ignored during the *DFU*.

Not all combinations of Bootloader, SoftDevice, and Application are possible when generating a package. See Table 1: Supported Bootloader, SoftDevice, and Application combinations on page 9 for more information.

4.1 DFU package combinations

The following table lists the supported combinations when generating a *DFU* package.

Combination	Supported
Bootloader (BL)	Yes[1]
SoftDevice (SD)	Yes[2]
Application (APP)	Yes
BL + SD	Yes
BL + APP	No[3]
BL + SD + APP	Yes[4]
SD + APP	Yes[2,4]

Table 1: Supported Bootloader, SoftDevice, and Application combinations

[1]Use nRF Util v5.0.0 or later when creating update packages of bootloaders compiled from nRF5 SDK 15.3.0 or later to ensure the correct size of generated packages.

Update packages of external applications (e.g. updates for third-party applications) are generated with the `--external-app` option. When this option is set, the receiving device stores the update but will not activate it. This functionality is experimental in the nRF5 SDK and is not yet used in any examples.

The `--zigbee` boolean option generates the Zigbee update file in addition to the zip package. For example:

```
nrfutil pkg generate --hw-version 52 --sd-req 0 --application-version 0x01020101
 --application nrf52840_xxaa.hex --key-file ../priv.pem --app-boot-validation
 VALIDATE_ECDSA_P256_SHA256 app_dfu_package.zip --zigbee True --zigbee-manufacturer-id
 0xCAFE --zigbee-image-type 0x1234 --zigbee-comment good_image --zigbee-ota-hw-version 52
 --zigbee-ota-fw-version 0x01020101 --zigbee-ota-min-hw-version 52 --zigbee-ota-max-hw-
version 52
```

The `--zigbee-ota-hw-version` and `--zigbee-ota-fw-version` options refer to the generated image to be distributed to the Zigbee OTA Server and disseminated later into the network. The `--zigbee-ota-hw-version` and `--zigbee-ota-fw-version` options describe the hardware version and firmware version of the Zigbee OTA Server respectively. Each time the Zigbee OTA Server receives the image for the dissemination, its firmware version is updated. This is done to protect from a malicious attack, where an attacker could wear down the Server flash memory by repeatedly sending the full Zigbee image to be distributed. Thus, in order for the OTA Server to accept the image for dissemination, the value passed as a `--zigbee-ota-fw-version` has to be incremented with every transfer of the image.

The `--zigbee-ota-min-hw-version` and `--zigbee-ota-max-hw-version` options refer to the fields in the Zigbee OTA header, which determine the range of the OTA Client's hardware version for which the image is suitable. Both `--zigbee-ota-min-hw-version` and `--zigbee-ota-max-hw-version` are optional and if used, both must be given.

[2]The SD must be of the same Major Version as the old BL may not be compatible with the new SD.

[3]Create two ZIP packages.

[4]The SD (+ BL) + APP update is done with two consecutive connections, unless a custom bootloader is used. First the SD (+ BL) is updated, then the bootloader will disconnect and the (new) BL will start advertising. Then the second connection to the bootloader will update the APP.

However, the two SDs may have different IDs. The first update requires `--sd-req` to be set to the ID of the old SD. The APP update requires the ID of the new SD. The new ID must be set using `--sd-id`

parameter. This parameter was added in nRF Util v3.1.0 and is required since v3.2.0 in case the package should contain SD (+ BL) + APP. Also, the new ID is copied to `--sd-req` list so that in case of a link loss during the APP update the *DFU* process can be restarted. In this case, the new SD would overwrite itself, so `--sd-req` must contain the ID of the new SD.

5 Performing a DFU

The **dfu** command transfers a *DFU* package to the nRF5 device.

There are several *DFU* transports available. Enter the following command to display a list of supported transports:

```
nrfutil dfu --help
```

Make sure that the transport you select matches the DFU bootloader that is installed on the DFU target device.

5.1 DFU over Bluetooth LE

Use an nRF5 *Development Kit (DK)* as the connectivity device for the *DFU* over Bluetooth LE procedure.

Complete the following steps to do the *DFU*:

1. Connect an nRF5 *DK* to your computer.

   Note: In the `-ic` option, you must specify if the *DK* contains an nRF51 or nRF52 chip.

2. Run **`nrfjprog --eraseall`** to erase the contents of the *DK*.
3. Run **`nrfutil dfu ble`** to do a full *DFU* over Bluetooth LE.

   Use the `-f` option to program the *DK* with the required connectivity software. This option overwrites the contents of the *DK*.

   Enter **`nrfutil dfu ble --help`** to see available options.

For example, to perform a *DFU* procedure using `app_dfu_package.zip` over Bluetooth LE on an nRF52 device connected to COM3, where MyDevice is the remote Bluetooth LE device being upgraded, enter the following command:

```
nrfutil dfu ble -ic NRF52 -pkg app_dfu_package.zip -p COM3 -n "MyDevice" -f
```

5.2 DFU over ANT

Use an ANT USB dongle (ANT USB-m for example) as the connectivity device for the *DFU* over ANT procedure.

Complete the following steps to perform the *DFU*:

1. Connect an ANT USB dongle to your computer.
2. Run **`nrfutil dfu ant`** to do a full *DFU* over ANT.

   You can see available options by entering the following command:

   ```
   nrfutil dfu ant --help
   ```

For example, enter the following command to perform a DFU procedure on the `app_dfu_package.zip` file:

```
nrfutil dfu ant -pkg app_dfu_package.zip
```

5.3 DFU over Thread

Use an nRF5 *DK* as the connectivity device for the *DFU* over Thread procedure.

For *DFU* over Thread, nRF Util supports both unicast and multicast mode. In unicast mode, every *DFU* client requests consecutive blocks of firmware from nRF Util individually. In multicast mode, nRF Util sends consecutive blocks of firmware in multicast messages, and clients that are interested in a new firmware receive and process these messages.

The default mode is unicast mode. To select multicast mode, call nRF Util with a multicast address as a destination address.

Complete the following steps to perform the *DFU*:

1. Connect an nRF5 *DK* to your computer.

   This board serves as the Thread network co-processor (NCP) for performing the *DFU* on the target.
2. Run **`nrfjprog --eraseall`** to erase the contents of the *DK*.
3. Run **`nrfutil dfu thread`** to do a full *DFU* over a Thread.

   Use the `-f` option to program the *DK* with the required connectivity software. This option overwrites the contents of the *DK*.

   Enter **`nrfutil dfu thread --help`** to see available options.

For example, enter the following command to initiate a unicast DFU procedure for the file `app_dfu_package.zip` over Thread on channel 11 with PAN ID 0xABCD, using an nRF52840 NCP connected to COM3:

```
nrfutil dfu thread -f -pkg app_dfu_package.zip -p COM3 --channel 11 --panid 43981
```

Any remote Thread device can then request the firmware update.

Enter the following command to perform a multicast DFU procedure for the file `app_dfu_package.zip` over Thread on channel 11 with PAN ID 0xABCD to the multicast address FF03::1, using an nRF52840 NCP connected to COM3:

```
nrfutil dfu thread -f -pkg app_dfu_package.zip -p COM3 --channel 11 --panid 43981 -r 4 -rs
 5000 -a FF03::1
```

Any remote Thread device can then decide whether it wants to receive and process the firmware update messages.

5.4 DFU over Zigbee

Use an nRF5 *DK* as the connectivity device for the *DFU* over Zigbee procedure.

Before you begin, run the following command to generate a Zigbee-specific image from your own application :

```
nrfutil pkg generate --hw-version 52 --sd-req 0x00 --application-version 0x01020101
 --application app.hex --key-file key.pem app_dfu_package.zip --zigbee True --zigbee-
manufacturer-id 123 --zigbee-image-type 321 --zigbee-comment good_image
```

The *DFU* over Zigbee procedure is performed by loading an upgrade image to the OTA Server running on the *DK*.

Complete the following steps to perform the *DFU*:

1. Connect an nRF5 *DK* to your computer.

This board serves as the Zigbee OTA Server which distributes a Zigbee image in the network.

2. Run **`nrfutil dfu zigbee`** to start the Zigbee OTA Server which is going to distribute new firmware in the network.

   Enter **`nrfutil dfu zigbee --help`** to see available options.

The *DFU* over Zigbee process ends immediately after loading the image, but the OTA Server is active until *DK* reset.

For example, enter the following command to start the Zigbee OTA Server that distributes the file `CAFE-1234-good_image.zigbee` on channel 20, using an nRF52840 *DK* with serial number 683604699:

```
nrfutil dfu zigbee -f CAFE-1234-good_image.zigbee -snr 683604699 -chan 20
```

5.4.1 Updating external applications

Packages for updating external applications can be generated by nRF Util by setting the `–external-app` option.

This is only available for updates that contain an application and no SoftDevice or bootloader. Setting this option instructs the receiving device that the update should be stored, but not activated, and then passed on to a third party. The following command is an example for generating an external application update package:

```
nrfutil pkg generate --hw-version 52 --application-version 0x01020101 –application app.hex
 --key-file key.pem app_dfu_package.zip --zigbee True --zigbee-manufacturer-id 123 --
zigbee-image-type 321 --zigbee-comment good-image --external-app --zigbee-ota-hw-version
 231
```

Note: This functionality is experimental in the nRF5 SDK and not used in any current examples.

5.5 DFU over a serial UART connection

The nRF Util tool supports *DFU* over a serial UART connection.

Complete the following steps to perform the *DFU*:

1. Connect the *DFU* target to your computer.

   Most Nordic Semiconductor *DK*s have an interface MCU that serves as a virtual COM port and transparently maps the UART into a USB CDC ACM interface. See Virtual COM port for more information. If no interface MCU is available, use other options to connect the *DFU* target to your computer, such as a USB to TTL adapter or a serial cable with an RS-232 connector.

2. Run **`nrfutil dfu serial`** to do a full *DFU* over a serial UART connection.

   You can see available options by entering the following command:

   ```
   nrfutil dfu serial --help
   ```

For example, enter the following command to perform a DFU procedure of the file `app_dfu_package.zip` over COM3 at 115200 bits per second:

```
nrfutil dfu serial -pkg app_dfu_package.zip -p COM3 -b 115200
```

5.6 DFU over a serial USB connection

The nRF Util tool supports *DFU*s over a USB CDC ACM connection.

DFU over a serial USB connection is supported only for chips that have USB pins (for example, nRF52840).

> **Note:** The USB port for the interface MCU is not connected to the USB pins on the chip. If you are using the interface MCU, you must perform a DFU over a serial UART connection.

Complete the following steps to perform the *DFU*:

1. Connect the *DFU* target to your computer.

   If your *DFU* target is an nRF52840 *DK*, use the USB port marked **nRF USB**.
2. Run **`nrfutil dfu usb-serial`** to do a full *DFU* procedure over a USB CDC ACM connection.

   You can see available options by entering the following command:

   ```
   nrfutil dfu usb-serial --help
   ```

For example, enter the following command to perform a DFU procedure of the file `app_dfu_package.zip` over COM3 at 115200 bits per second:

```
nrfutil dfu usb-serial -pkg app_dfu_package.zip -p COM3 -b 115200
```

6 Generating and displaying keys

The **keys** command can be used to generate and display cryptographic keys.

Cryptographic keys are required to sign and validate a *DFU* package. See the Cryptography library in the SDK for more information about signing and cryptographic keys.

- Run **nrfutil keys generate** to generate a private (signing) key and store it in a file in PEM format.
- Run **nrfutil keys display** to display a private (signing) or public (verification) key from a PEM file.

  You can see available options by entering the following command:

  ```
  nrfutil keys display --help
  ```

For example, enter the following command to generate a private key and store it in a file named `private.pem`:

```
nrfutil keys generate private.pem
```

Enter the following command to display a public key in code format from this key file:

```
nrfutil keys display --key pk --format code private.pem
```

7 Generating and displaying bootloader settings

Use the **`settings`** command to generate and display a bootloader settings page.

A *DFU* bootloader requires a bootloader settings page that contains information about the current *DFU* process. In addition, it can contain information about the installed application and the firmware version.

- Run **`nrfutil settings generate`** to generate a bootloader settings page and store it in a HEX file.

  You can see available options by entering the following command:

  ```
  nrfutil settings generate --help
  ```

- Run **`nrfutil settings display`** to display the contents of a bootloader settings page that is present in a HEX file.

  To read the bootloader settings page from a programmed device, use **`nrfjprog`** to dump the flash memory of the IC (where *HEX_file* is the name of the resulting HEX file):

  ```
  nrfjprog --readcode HEX_file
  ```

After generating the bootloader settings page, you can use **`mergehex`** and **`nrfjprog`** to program it to the device. See the nRF Command Line Tools documentation for more information.

For example, enter the following command to generate a bootloader settings page for an nRF52840 device with the application `app.hex` installed, with application version 3, bootloader version 2, and bootloader settings version 1 (for SDK v13.0.0), and store it in a file named `settings.hex`:

```
nrfutil settings generate --family NRF52840 --application app.hex --application-version 3
 --bootloader-version 2 --bl-settings-version 1 settings.hex
```

Enter the following command to display the contents of the generated HEX file:

```
nrfutil settings display settings.hex
```

Each nRF device has a corresponding `--family` setting:

Family setting	nRF devices
NRF51	nRF51xxx
NRF52	nRF52832, nRF52833
NRF52QFAB	nRF52832-QFAB, nRF52820
NRF52810	nRF52810, nRF52811, nRF52805
NRF52840	nRF52840

Table 2: `--family` settings

The `--bl-settings-version` depends on the SDK version:

SDK version	BL settings version
12.0	1
15.3.0	2

Table 3: SDK and BL settings versions

The *DFU* bootloader settings version supported and used by your selected SDK is listed in the `nrf_dfu_types.h` file in the `bootloader` library. Even though bootloaders compiled from an nRF5 SDK 15.3.0 or later can only use version 2, they can be configured to support a version 1 settings page. When a new bootloader with a version 1 settings page boots, the bootloader translates the settings page to version 2 before booting. If a version 2 settings page is used, boot validation for SoftDevice and Application can be generated with the settings page using the **`--sd-boot-validation`** and **`--app-boot-validation`** commands.

8 Generating HEX files for Zigbee

Use the **`zigbee production_config`** command to generate production configuration HEX files for Zigbee devices.

Production configuration HEX files can be used to set the production configuration flash memory block in the Zigbee stack when working with the nRF5 SDK for Thread and Zigbee. See the Zigbee production configuration section in nRF5 SDK for Thread and Zigbee for more information.

1. Create a `yaml` input file that contains the list of required values that are used to generate the production configuration HEX file.

   Enter the following command for an example of the yaml input and other available options:

   ```
   nrfutil zigbee production_config --help
   ```

2. Run **`nrfutil zigbee production_config`** and provide input and output file names to generate a binary file in Intel HEX format.

For example, enter the following command to generate a production configuration `output.hex` file from the `input.yaml` file:

```
nrfutil zigbee production_config input.yaml output.hex
```

9 Displaying version information

The **`version`** command can be used to display the version of the tool.

Different versions of nRF Util support different formats of the init packet that is part of the DFU package. Use the nRF Util version that corresponds to the DFU bootloader that is programmed on your device.

Enter the following command to display the nRF Util version:

```
nrfutil version
```

10 Customizing the init packet

The init packet is a packet that is sent before the actual firmware images in a *DFU*. It contains metadata about the *DFU*, such as the size and type of the image, version information, and compability requirements.

To customize the tool, you must have cloned the nRF Util GitHub repository (see Installing from sources on page 5).

If you use the default packet format as described in the BLE Secure DFU Bootloader example, you do not need to modify nRF Util. If you define a custom init packet format, however, you must modify both your DFU bootloader implementation and nRF Util to use this new format.

The format of the init packet is defined in a Protocol buffers (.proto) file. This file can be compiled into different formats, ensuring that you use the same init packet format in you DFU bootloader implementation and in nRF Util.

Note: The init packet definition requires the *proto2* version of the protocol buffers language. Do not include `syntax = "proto3";` in your protocol buffer file, because this would cause the file to be interpreted as a *proto3* language version file.

Complete the following steps to customize the init packet:

1. Modify the protocol buffer file to suit your needs.

   In the nRF Util GitHub repository, the file is located at `nordicsemi/dfu/dfu-cc.proto`. In the BLE Secure DFU Bootloader example in the nRF5 SDK, it is located at `examples/dfu/bootloader_secure/dfu-cc.proto`. Ensure that both files have the exact same content.
2. Download and install the Protocol compiler from Google.
3. Adapt nRF Util to use the new init packet format:
   a) Compile the protocol buffer file to generate the corresponding Python file.

      In the folder that contains your `dfu-cc.proto` file, enter the following command (where *dest_folder* is an empty folder where the protocol compiler will write its output):

      ```
      protoc --python_out=dest_folder dfu-cc.proto
      ```

   b) Copy or move the created file *dest_folder*`/dfu_cc_pb2.py` to `nordicsemi/dfu/`, overwriting the existing file.
   c) If you added new information to the init packet you need to update nRF Util. Adapt nRF Util to include command-line options for new fields and add the information provided through these options to the init packet.

      To adapt the tool, you must edit the Python source files. The contents of the init packet are defined in the files `nordicsemi/dfu/init_packet_pb.py` and `nordicsemi/dfu/package.py`. The command-line options are defined in `nordicsemi/__main__.py`.

   If you installed nRF Util to the local Python site-packages and script folder or created a self-contained executable, you must repeat that procedure after editing the source files.
4. Adapt your DFU bootloader implementation to use the new init packet format:
   a) Make sure that the `dfu-cc.proto` file in the request handling folder of your DFU bootloader implementation (by default, this is the `examples/dfu/dfu_req_handling/` folder in the SDK) is the same file that you used to adapt nRF Util.
   b) In the DFU bootloader implementation folder, enter the following command:

      ```
      protoc -odfu-cc.pb dfu-cc.proto
      ```

      This command creates the file `dfu-cc.pb`.

c) If you are not working in the `examples/dfu/dfu_req_handling/` folder, copy the `dfu-cc.options` file from that folder to your implementation folder.

d) Run the `nanopb_generator.py` script (located in the `external/nano-pb/generator/` folder) to generate the required `dfu-cc.pb.c` and `dfu-cc.pb.h` files.

   If you are working in `examples/dfu/dfu_req_handling/`, enter the following command:

   ```
   python ../../../external/nano-pb/generator/nanopb_generator.py dfu-cc.pb -f dfu-cc.options
   ```

   If you are working in a different folder, adapt the path to the script.

e) Compile your DFU bootloader.

For more information about this procedure, see the readme file in the BLE Secure DFU Bootloader folder in the SDK.

Glossary

Device Firmware Update (DFU)

A mechanism for upgrading the firmware of a device.

Development Kit (DK)

A development platform used for application development.

Acronyms and abbreviations

These acronyms and abbreviations are used in this document.

DFU
: Device Firmware Update

DK
: Development Kit

Legal notices

By using this documentation you agree to our terms and conditions of use. Nordic Semiconductor may change these terms and conditions at any time without notice.

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function, or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Nordic Semiconductor ASA does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. If there are any discrepancies, ambiguities or conflicts in Nordic Semiconductor's documentation, the Product Specification prevails.

Nordic Semiconductor ASA reserves the right to make corrections, enhancements, and other changes to this document without notice.

Life support applications

Nordic Semiconductor products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury.

Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

RoHS and REACH statement

Complete hazardous substance reports, material composition reports and latest version of Nordic's REACH statement can be found on our website www.nordicsemi.com.

Trademarks

All trademarks, service marks, trade names, product names, and logos appearing in this documentation are the property of their respective owners.

Copyright notice