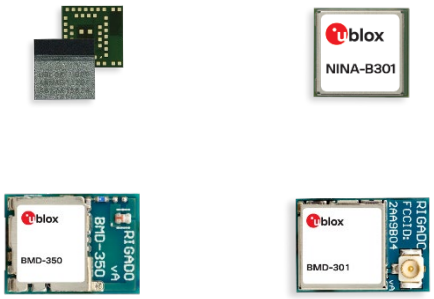
Add Nordic Semiconductor DFU to SDK example

Bluetooth low energy

Application note



Abstract

A device firmware update bootloader (DFU) provides a means of updating application firmware over the air (OTA) in the field. This application note describes the procedure to add a buttonless DFU bootloader to an existing application and perform an OTA update.

UBX-19050198 - R03
C1-Public

Document information

Title	**Add Nordic Semiconductor DFU to SDK example**	
Subtitle	Bluetooth low energy	
Document type	Application note	
Document number	UBX-19050198	
Revision and date	R03	8-Jan-2021
Disclosure restriction	C1-Public	

This document applies to the following products:

Product name	EVK
ANNA-B112	EVK-ANNA-B112
BMD-300	BMD-300-EVAL
BMD-301	BMD-301-EVAL
BMD-330	BMD-330-EVAL
BMD-340	BMD-340-EVAL
BMD-341	BMD-341-EVAL
BMD-345	BMD-345-EVAL
BMD-350	BMD-350-EVAL
BMD-360	BMD-360-EVAL
BMD-380	BMD-380-EVAL
NINA-B111	EVK-NINA-B111
NINA-B112	EVK-NINA-B112
NINA-B301	EVK-NINA-B301
NINA-B302	EVK-NINA-B302
NINA-B306	EVK-NINA-B306
NINA-B400	EVK-NINA-B400
NINA-B406	EVK-NINA-B406

u-blox or third parties may hold intellectual property rights in the products, names, logos and designs included in this document. Copying, reproduction, modification or disclosure to third parties of this document or any part thereof is only permitted with the express written permission of u-blox.
The information contained herein is provided “as is” and u-blox assumes no liability for its use. No warranty, either express or implied, is given, including but not limited to, with respect to the accuracy, correctness, reliability and fitness for a particular purpose of the information. This document may be revised by u-blox at any time without notice. For the most recent documents, visit www.u-blox.com.
Copyright © u-blox AG.

Contents

Document information **2**
Contents **3**
1 Software preparation **4**
1.1 Nordic Semiconductor nRF5 SDK 4
1.2 SEGGER Embedded Studio 4
1.3 Nordic utilities 4
1.3.1 nRF Connect for Desktop 4
1.3.2 nRF Command Line Tools 4
1.3.3 nRF Util 4
1.4 Encryption libraries 5
1.4.1 Git 5
1.4.2 GCC for ARM 5
1.4.3 GNU make 5
1.4.4 Fetch and compile the encryption libraries 6
2 Hardware preparation **7**
2.1 Bluetooth address 7
3 Application firmware **8**
4 Button DFU **12**
4.1 DFU bootloader 12
4.2 Key generation 12
4.3 Firmware update package generation 13
5 Buttonless DFU **19**
5.1 Preprocessor definitions 19
5.2 sdk_config.h 22
5.3 Libraries 22
5.4 main.c 22
5.5 Hex file generation 24
6 Test DFU **27**
6.1 Program and test application 27
6.2 Prepare and test update 28
Appendix **30**
A Glossary **30**
Related documents **31**
Revision history **31**
Contact **32**

1 Software preparation

1.1 Nordic Semiconductor nRF5 SDK

A fresh installation of the nRF5 SDK is assumed. The SDK may be downloaded from reference [6]. For this application note, the installation is assumed to be located in:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099
```

1.2 SEGGER Embedded Studio

Nordic Semiconductor have partnered with SEGGER to provide a no-cost, no-size-limit, commercial license for SEGGER Embedded Studio (SES) for use with Nordic ICs, including the nRF5x series on which many u-blox modules are based. Install SES from SEGGER and a license from Nordic:

SES: https://www.segger.com/downloads/embedded-studio/

License: https://license.segger.com/Nordic.cgi

1.3 Nordic utilities

Several utilities from Nordic Semiconductor are used throughout the process. Install these utilities:

1.3.1 nRF Connect for Desktop

nRF Connect for Desktop provides several tools for working with the Nordic Semiconductor-based u-blox modules. nRF Connect for Desktop may be downloaded from reference [7].

1.3.2 nRF Command Line Tools

nRF Command Line Tools provides `nrfjprog` and `mergehex`, used for development, programming, and debugging. nRF Command Line Tools may be downloaded from reference [8].

1.3.3 nRF Util

nRF Util is used for generating the DFU update packages, bootloader settings generation, key generation, and performing updates. nRF Util may be downloaded from the Nordic Semiconductor GitHub repository [9].

nRF Util depends on Python v3.7 or later. Python may be downloaded from reference [10].

☞ It is suggested to install the python component `wheel` before installing `nrfutil`:

```
pip install wheel
```

1.4 Encryption libraries

Encryption libraries are required to create the signed update files use with DFU. Follow the steps outlined in the Nordic instructions to add the encryption libraries to the SDK. For installation instructions, see reference [11].

The following installations are required to install the encryption libraries.

1.4.1 Git

Git is a distributed version control system used by many development projects. macOS supports git directly through installing X-Code. On Linux, install git with the package manager for the operating system. On Windows, install it from reference [12].

☞ Any other form of git may be used, such as through Linux installed on Windows through WSL or a virtual machine.

The SDK uses git to fetch the source code for micro-ecc.

1.4.2 GCC for ARM

The GCC for ARM compiler is used to compile the micro-ecc libraries. GCC for ARM may be downloaded from reference [13].

The nRF5 SDK v17 uses a newer version of the ARM GCC compiler than the instructions at the InfoCenter link indicates. Download this version:

GNU Arm Embedded Toolchain: 7-2018-q2-update June 27, 2018

☞ When executing the installer, be sure to select the option to "Add path to environment variable".

1.4.3 GNU make

The `make` utility is used to manage the build process. There are several ways to install it. macOS and Linux include `make` as part of the usual development tools. For Windows, the installer is found at reference [14].

On Windows, update the PATH environment variable to include the binary folder for `make.exe`:

```
C:\Program Files (x86)\GnuWin32\bin
```

1.4.4 Fetch and compile the encryption libraries

Once the GCC compiler, git, and make are in place, a shortcut provided within the SDK is used to do compile and install the encryption libraries. Open a Command window and navigate to the directory

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\external\micro-ecc
```

A batch file, and a script for macOS and Linux, called `build_all` is used to compile and install micro-ecc.

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\external\micro-ecc>build_all.bat
"micro-ecc not found! Let's pull it from HEAD."
Cloning into 'micro-ecc'...
remote: Enumerating objects: 1086, done.
remote: Total 1086 (delta 0), reused 0 (delta 0), pack-reused 1086
00 KiB/s
Receiving objects: 100% (1086/1086), 647.94 KiB | 921.00 KiB/s, done.
Resolving deltas: 100% (637/637), done.
make: Entering directory `C:\u-blox\nRF5_SDK_17.0.0_9d13099/external/micro-
ecc/nrf51_armgcc/armgcc'
mkdir _build
cd _build && mkdir micro_ecc_lib
Compiling file: uECC.c
...
...
ecc/nrf52nf_iar/armgcc'
make: Entering directory `C:\u-blox\nRF5_SDK_17.0.0_9d13099/external/micro-
ecc/nrf52nf_keil/armgcc'
mkdir _build
cd _build && mkdir micro_ecc_lib
Compiling file: uECC.c
Creating library: ../../nrf52nf_keil/armgcc/micro_ecc_lib_nrf52.lib
C:/Program Files (x86)/GNU Tools ARM Embedded/7 2018-q2-update/bin/arm-none-eabi-ar:
creating ../../nrf52nf_keil/armgcc/micro_ecc_lib_nrf52.lib
Done
make: Leaving directory `C:\u-blox\nRF5_SDK_17.0.0_9d13099/external/micro-
ecc/nrf52nf_keil/armgcc'

C:\u-blox\nRF5_SDK_17.0.0_9d13099\external\micro-ecc>
```

At this point, the SDK has the necessary encryption libraries not only for DFU use, but also for establishing secure connections through pairing and bonding.

The preparation to this point only needs done once after the SDK zip file is extracted.

2 Hardware preparation

For this application note, the BMD-340-EVAL board and a nRF52840 USB dongle are used to demonstrate the process.

Any of the nRF52840 or nRF52832 boards may be used: EVK-NINA-B3, nRF52840 Dongle, nRF52840 DK, EVK-ANNA-B1, EVK-NINA-B1, BMD-300-EVAL, BMD-301-EVAL, BMD-350-EVAL, or Nordic Semiconductor nRF52 DK.

If an alternative board is being used, adjust the directories for the project:

nRF52840 / BMD-340-EVAL, BMD-341-EVAL, BMD-345-EVAL, BMD-380-EVAL, EVK-NINA-B3

- S140 SoftDevice
- PCA10056 hardware directory
- `custom_board.h` for NINA-B3, BMD-345

☞ When working with the BMD-345, be sure to properly configure the PA / LNA in both the bootloader and application. This information is found in the BMD-345 data sheet [4].

nRF52832 / BMD-300-EVAL, BMD-301-EVAL, BMD-350-EVAL, EVK-ANNA-B1, EVK-NINA-B1

- S132 SoftDevice
- PCA10040 hardware directory
- `custom_board.h` for ANNA-B1, NINA-B1

nRF52810 / BMD-330-EVAL

- S112 SoftDevice
- PCA10040e hardware directory

nRF52811 / BMD-360-EVAL

- S113 SoftDevice
- PCA10056e hardware directory

☞ The BMD-330-EVAL and BMD-360-EVAL have limited DFU functionality. For example, updating the SoftDevice (Bluetooth stack) is not supported due to the flash and RAM memory sizes.

2.1 Bluetooth address

u-blox modules are programmed with a unique, public Bluetooth address (MAC address).

⚠ Some of the activities outlined in this application note require fully erasing the module, including the Bluetooth address.

To save this address for later reference, use nrfjprog to read the User Information Configuration Register (UICR) area. This example uses the file name `evk_uicr.hex` to store the UICR contents:

```
nrfjprog --readuicr evk_uicr.hex
```

Save the hex file in a convenient location.

The UICR may be restored with:

```
nrfjprog --program evk_uicr.hex
```

The example does not use the public Bluetooth address. Instead, it uses a static random Bluetooth address that is found in the Factory Information Configuration Registers (FICR). Application note [5] describes how to add code to read and assign the public Bluetooth address held in the UICR.

3 Application firmware

Navigate to the example application called ble_app_uart:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart>
```

To maintain the integrity of the original example code, make a copy of the project and all its subdirectories. Only the content of the working directory will be modified:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working>
```

The BMD-340-EVAL board is functionally equivalent to the nRF52840 DK. Throughout the SDK, this board is referred to by its board number, "pca10056".

Navigate to the SES folder and open the project:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working\pca10056\s140\ses\ble_app_uart_pca10056_s140.emProject
```

Compile and load the application to the BMD-340-EVAL to confirm operation before any changes are mode. Details for testing the functionality are located at reference [15].

Open the file in nRF Connect to have a look at the memory layout of the hex file. The hex file is located in the Output directory for the example:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working\pca10056\s140\ses\Output\Release\Exe\ble_app_uart_pca10056_s140.hex
```

The Programmer app of nRF Connect is used to view the memory layout.

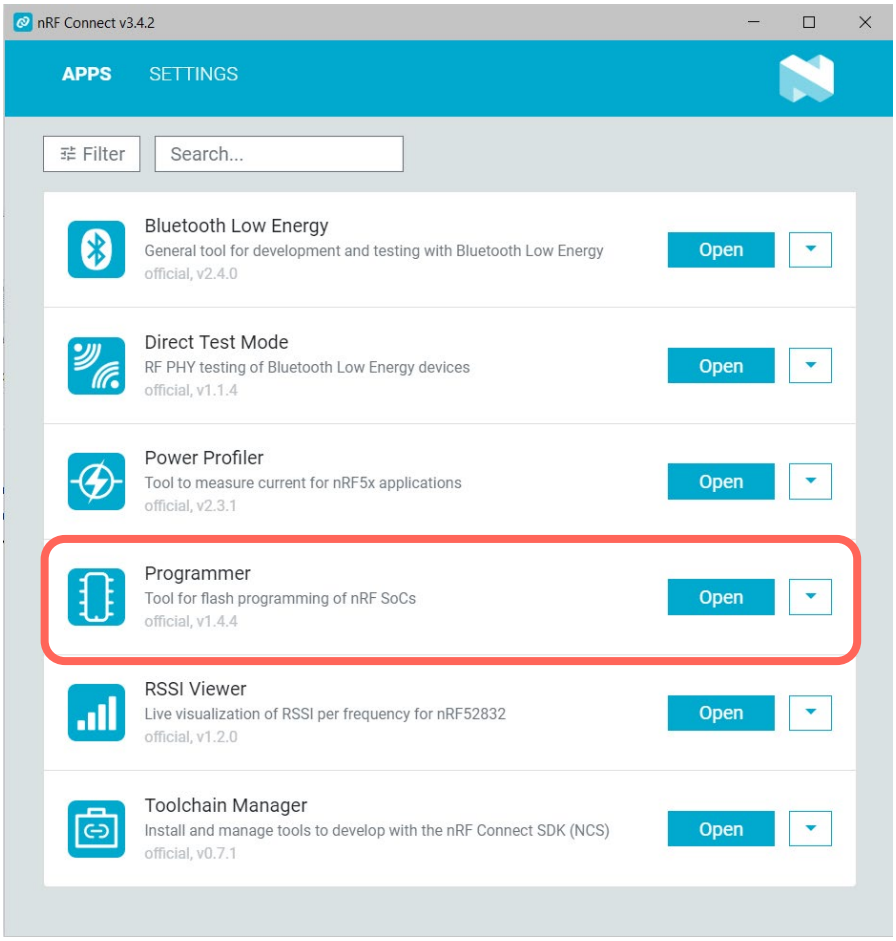


Figure 1: nRF Connect main window (programmer)

Add the hex file, or just drag it into the "File Memory Layout" section:

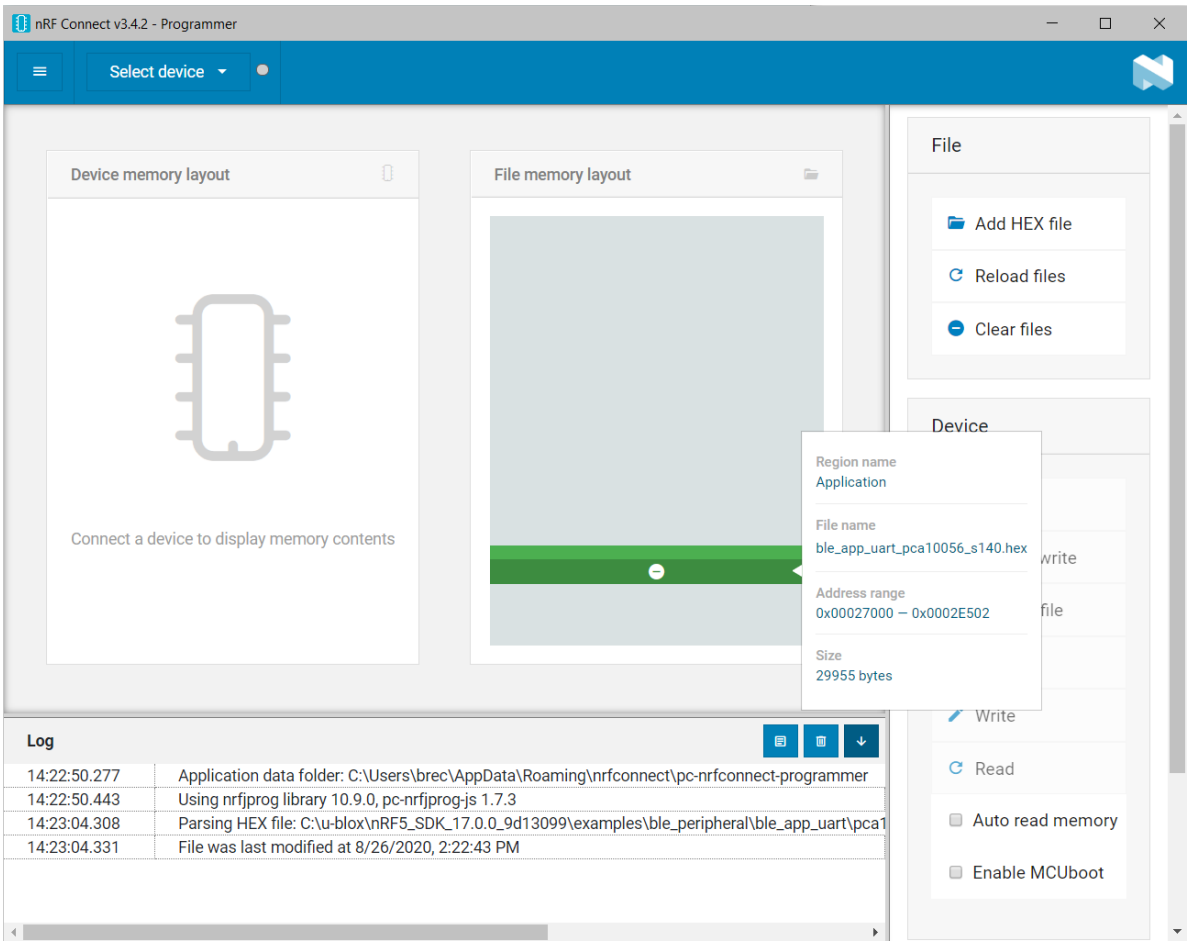


Figure 2: nRF Connect showing application only

Only the application itself is in the file. Notice the gap below. This is where the SoftDevice gets loaded. Add or drag the SoftDevice hex file to nRF Connect. The SoftDevice is located here:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\components\softdevice\s140\hex\s140_nrf52_7.0.1_softdevice.hex
```

Now both are shown, along with the MBR:

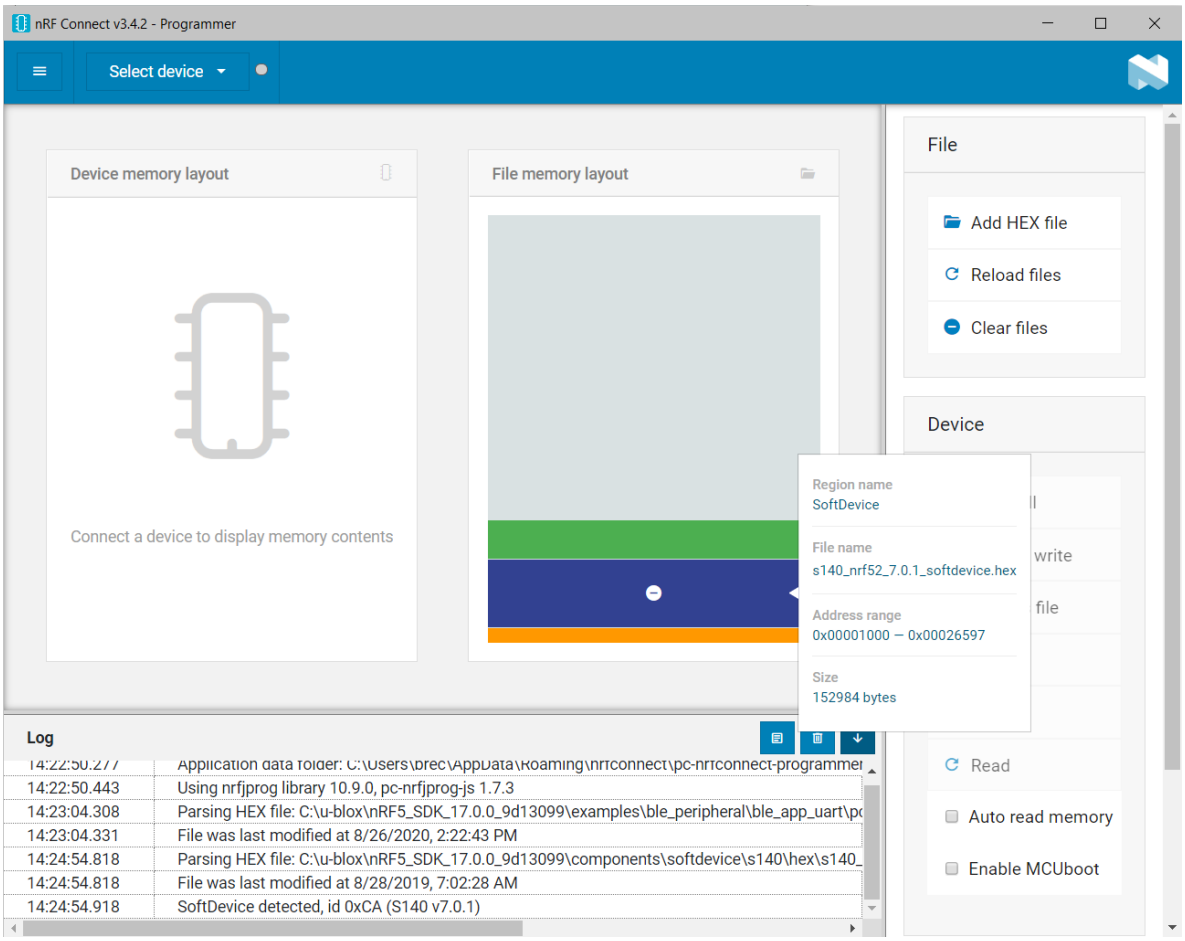


Figure 3: nRF Connect showing application with SoftDevice

While nRF Connect can be used to program the BMD-340, switch back to SES and load the SoftDevice and application from there:

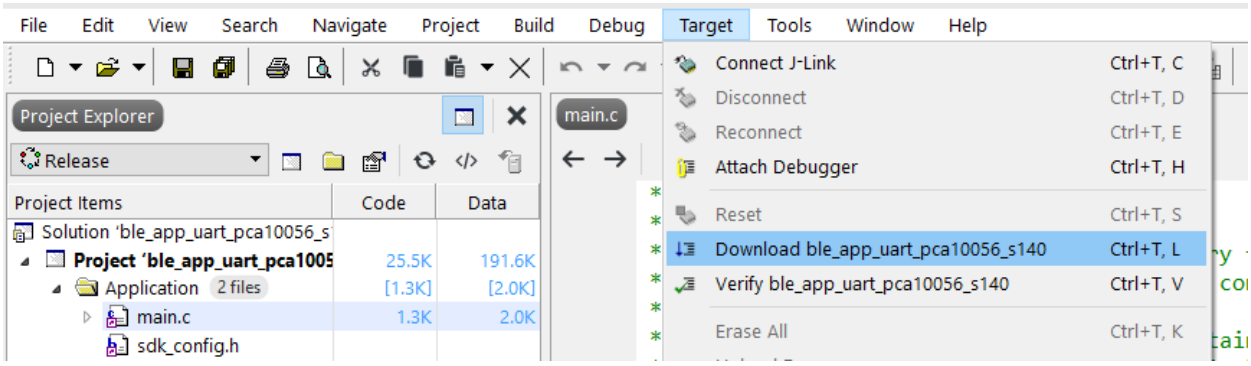


Figure 4: SES download application to target

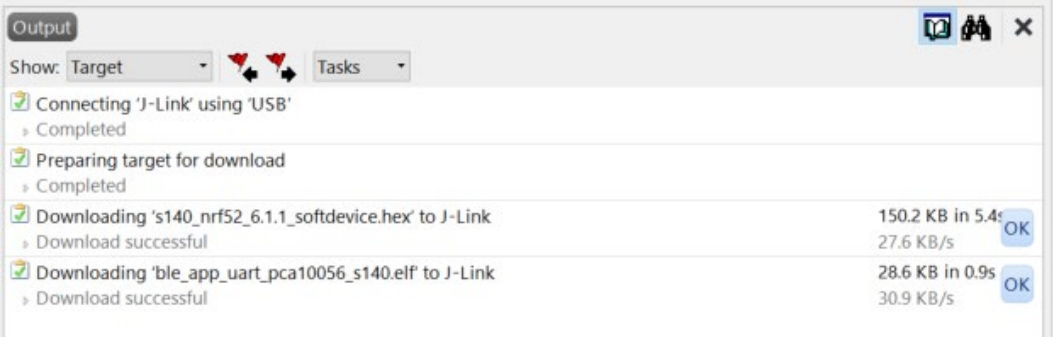


Figure 5: SES download progress window

LED1 should start flashing, indicating that the BMD-340 is advertising for a connection. Go ahead and test the application according to the instructions at the InfoCenter link above.

4 Button DFU

4.1 DFU bootloader

After confirming operation, the bootloader can now be added. Nordic Semiconductor provides several DFU examples. The Secure Bootloader example is used here. See the Nordic Semiconductor InfoCenter for details about this secure bootloader [16].

On the local drive, navigate to the `secure_bootloader` DFU directory of the examples and make a copy of the directory. Copy

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\secure_bootloader\pca10056_s140_ble
```

to

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\secure_bootloader\pca10056_s140_ble_working
```

Open the SES project for the pca10056 Bluetooth low energy bootloader:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\secure_bootloader\pca10056_s140_ble_working
\ses\secure_bootloader_ble_s140_pca10056.emProject
```

Compiling the bootloader “as is” will generate an error:

```
#error "Debug public key not valid for production. Please see
https://github.com/NordicSemiconductor/pc-nrfutil/blob/master/README.md to generate it"
```

⚠ Nordic Semiconductor supplies an example keys along with test files. These are only made available so the process of working with updating a file on a device can be tested; however, they cannot be used for production.

4.2 Key generation

`nrfutil` is used to generate a custom set of keys. Navigate to the directory where the default public key is located and change the name of the original key file, `dfu_public_key.c`:

```
move dfu_public_key.c dfu_public_key.c.NORDIC
```

Generate new keys. First, start with the private key:

```
nrfutil keys generate dfu_private_key.pem
Generated private key and stored it in: dfu_private_key.pem
```

Then create the public key in C language:

```
nrfutil keys display --key pk --format code --out_file dfu_public_key.c
dfu_private_key.pem
/* This file was automatically generated by nrfutil on 2019-06-21 (YY-MM-DD) at 14:25:09 */

#include "stdint.h"
#include "compiler_abstraction.h"

/** @brief Public key used to verify DFU images */
__ALIGN(4) const uint8_t pk[64] =
{
 0xc8, 0xc3, 0x75, 0xef, 0x51, 0x32, 0x99, 0x4a,
 0x24, 0xce, 0x53, 0xeb, 0x66, 0x2e, 0x49, 0x18,
 0xfd, 0x36, 0xa8, 0xff, 0x45, 0x3f, 0xbc, 0xa9,
 0x12, 0xdd, 0x03, 0xf7, 0x83, 0xbd, 0x71, 0x96,
 0xd8, 0xa7, 0x08, 0x32, 0xf9, 0x73, 0xd2, 0x9b,
 0xcd, 0xe7, 0x93, 0x13, 0x66, 0x38, 0x72, 0x8a,
 0x86, 0x58, 0x1a, 0x63, 0x33, 0x2c, 0x6f, 0x50,
 0x61, 0x68, 0x04, 0x6c, 0x1c, 0x9e, 0xf0, 0xbd
}; // line breaks added for readability
```

☞ These are example values. The actual output will vary.

The DFU project in SES can now be compiled without errors.

Fully erase the BMD-340-EVAL before proceeding:

```
nrfjprog -f nrf52 --recover
Recovering device. This operation might take 30s.
Erasing user code and UICR flash areas.
```

From the SES project, load the DFU and SoftDevice using the same procedure as the UART project.

Both LED1 and LED2 will light up solid. This indicates the BMD-340 is in bootloader mode. The module will be advertising with the name “DfuTarg”.

4.3 Firmware update package generation

The firmware update package consists of a signed zip file containing the application, SoftDevice, bootloader, and bootloader settings.

Navigate to the output directory of the ble_app_uart example that was compiled earlier.

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working\pca10056\s140\ses\Output\Release\Exe
```

`nrfutil` will be used with the package generate function. The help output will show all of the available optoins.

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

For this example, the following options are used:

Option	Description
`--application ble_app_uart_pca10056_s140.hex`	Compiler output hex file
`--application-version-string "1.2.3"`	Text version string The string “1.2.3” is converted to an integer: 10203
`--hw-version 52`	The default number "52" is used for a nRF52xxx device. This can actually be any integer as well, for example to reflect the host board version. If new code cannot be loaded onto an older version board, then this is the value for that check.

`--sd-req 0xCA`	If your application requires a specific version of the SoftDevice, this value is used for that check.
	☞ `nrfutil pkg generate -help` will list the available SoftDevice versions and their corresponding firmware IDs. More than one may be included here, comma separated.
	☞ The following SoftDevice v7.x.x codes are not listed in the help output, but can be used with `nrfutil`:
	`|s112_nrf52_7.0.0|0xC4|`
	`|s112_nrf52_7.0.1|0xCD|`
	`|s113_nrf52_7.0.0|0xC3|`
	`|s113_nrf52_7.0.1|0xCC|`
	`|s132_nrf52_7.0.0|0xC2|`
	`|s132_nrf52_7.0.1|0xCB|`
	`|s140_nrf52_7.0.0|0xC1|`
	`|s140_nrf52_7.0.1|0xCA|`

`--key-file C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\dfu_private_key.pem`	Path and file name of the new private .pem key
`app_v1.zip`	Output file name

Table 1: nrfutil package generation options

Putting it all together:

```
nrfutil pkg generate --application ble_app_uart_pca10056_s140.hex --application-version-string "1.2.3" --hw-version 52 --sd-req 0xCA --key-file C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\dfu_private_key.pem app_v1.zip
Zip created at app_v1.zip
```

To test this package, connect a second EVK or nRF52840 Dongle to your computer. Open nRF Connect and select the “Bluetooth Low Energy” option:

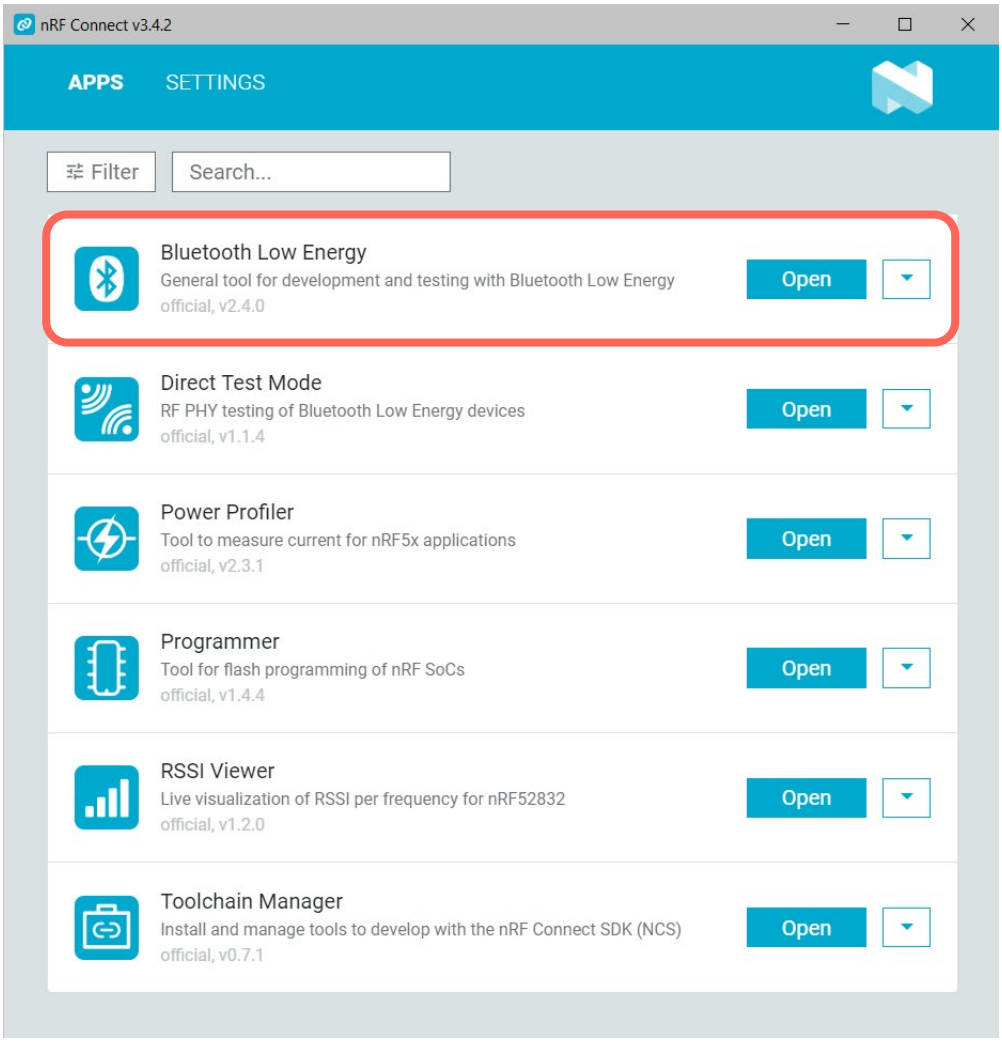


Figure 6: nRF Connect main window (Bluetooth Low Energy)

In this example the nRF52840 dongle is the second device. Select this device.

☞ If a notice to program the connectivity firmware is displayed, select yes.

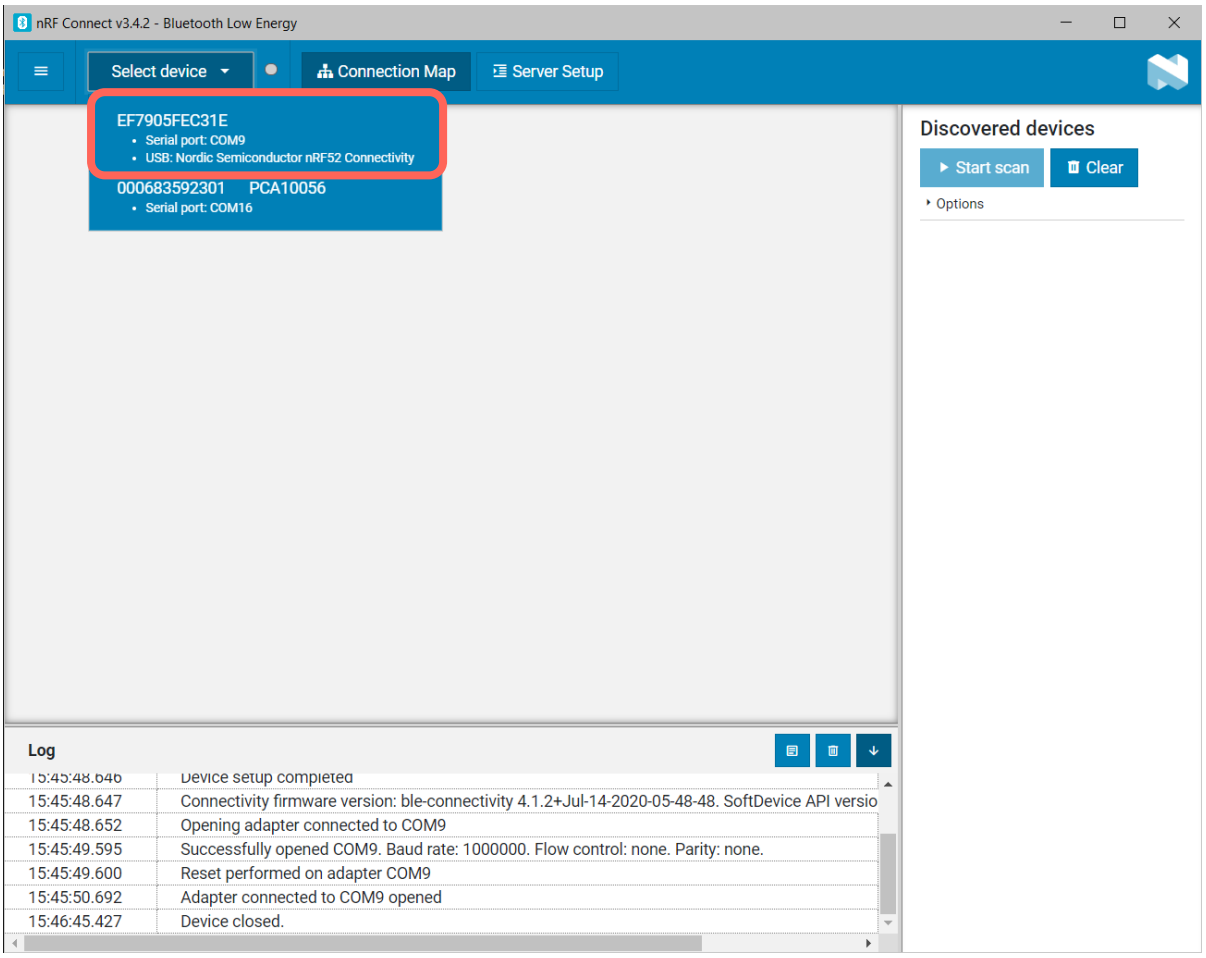


Figure 7: nRF Connect Bluetooth Low Energy window

Start the scan and look for the Eval board advertising "DfuTarg":

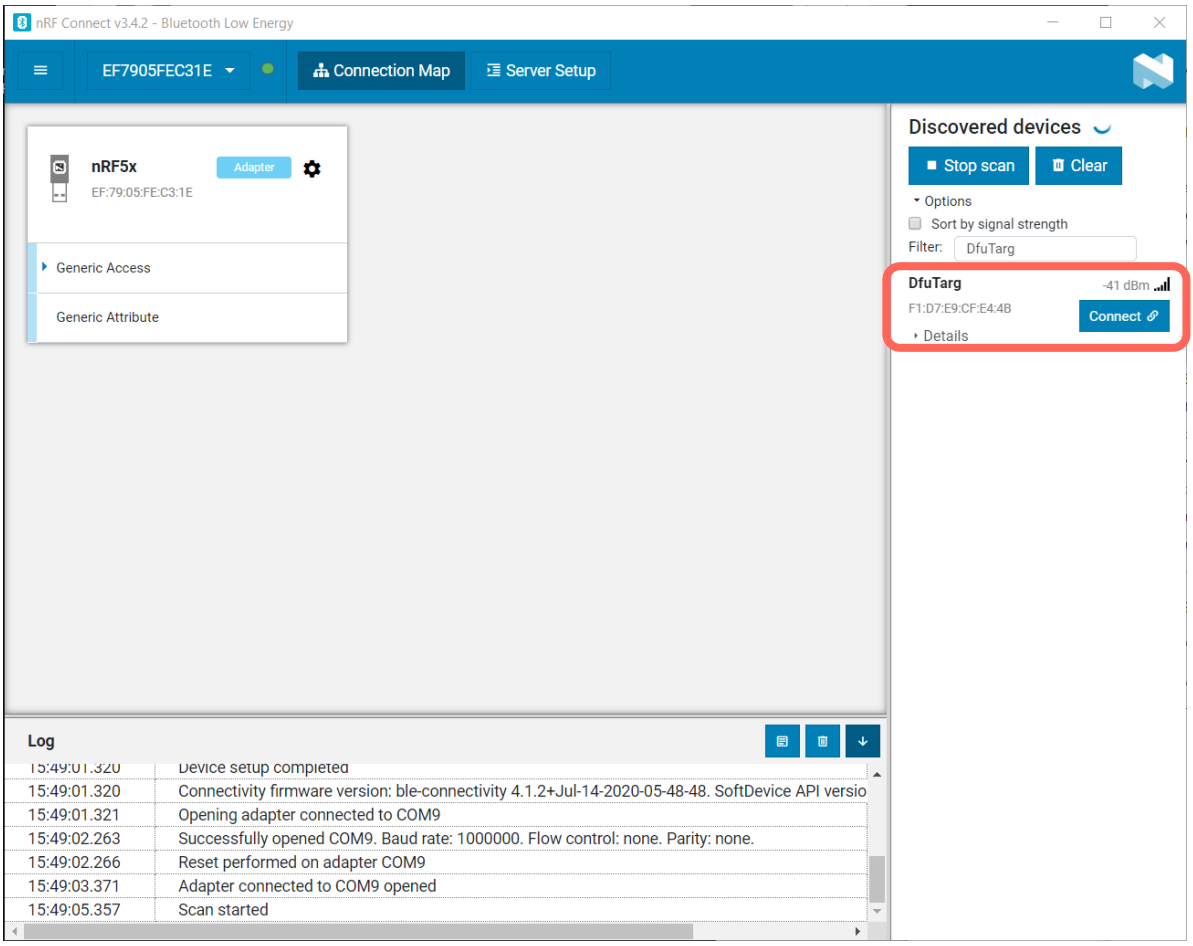


Figure 8: nRF Connect showing scanned bootloader DfuTarg

Select “Connect”. Observe the "Secure DFU" service and update icon:

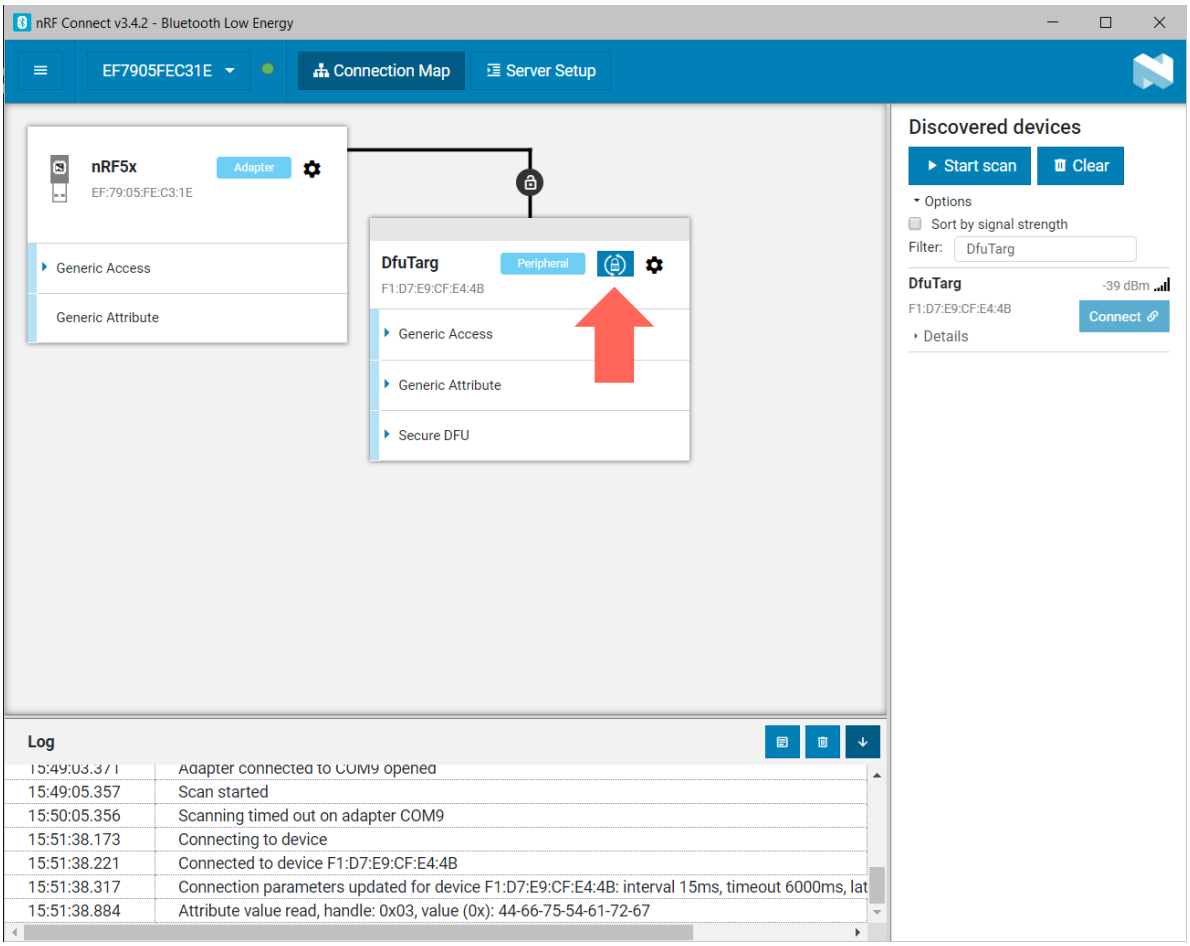


Figure 9: nRF Connect showing secure DFU icon

Select the icon to start the DFU process. Navigate to the zip file, then select "Start DFU".

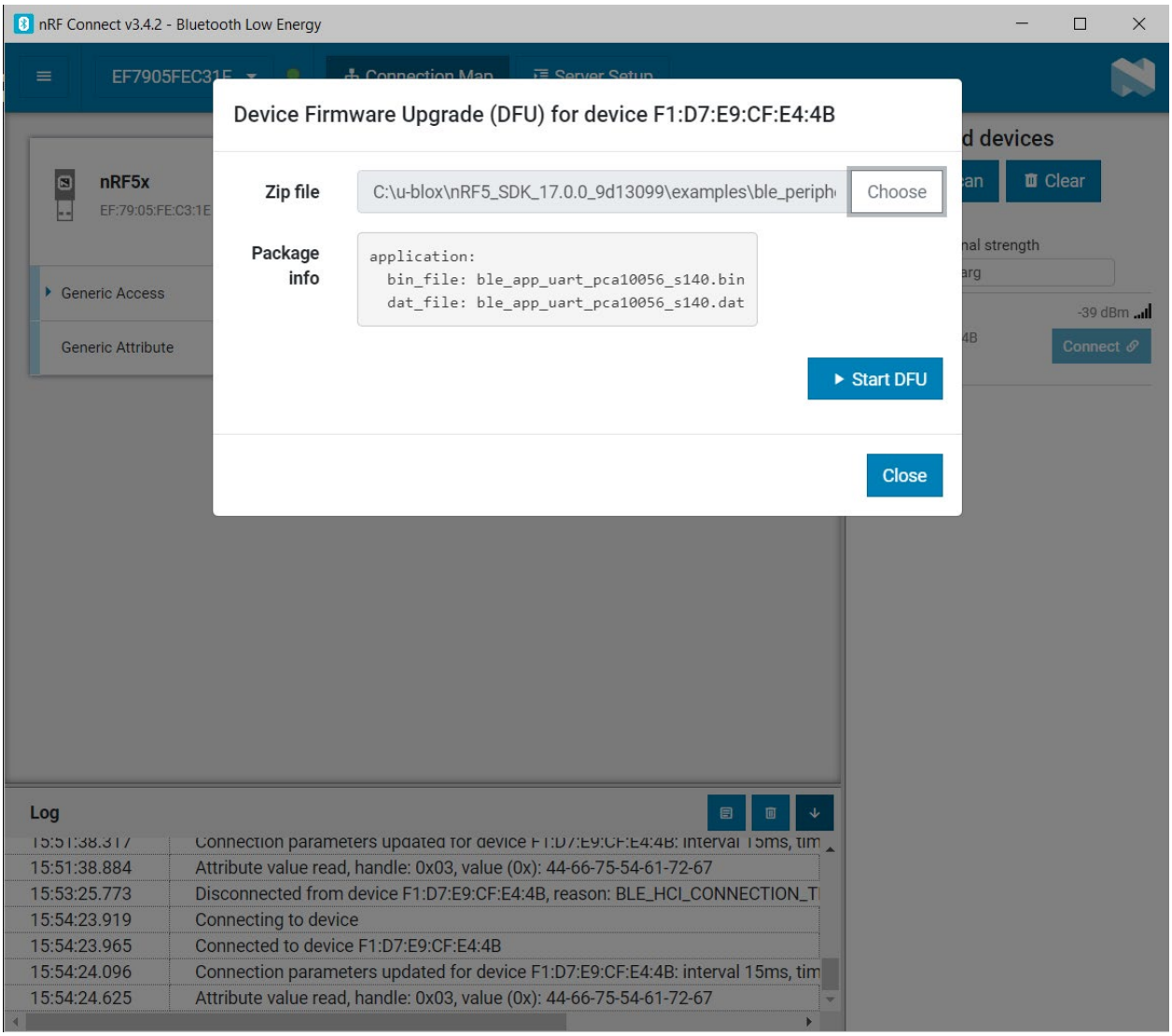


Figure 10: nRF Connect ready for DFU upgrade

A status bar will show up, first with a progress bar, then "Complete".

At this point, the Eval board will start advertising "Nordic_UART" – the example from Section 3 – though now it includes the bootloader.

If power to the Eval board is reset or power-cycled while holding Button 4, it will re-enter bootloader mode indicated by both LED1 and LED2 being lit.

Up to this point, the following is completed:

- Confirmed that the application code works as expected
- Generated private and public keys for the bootloader and update files
- Compiled the bootloader with the new public key
- Generated a DFU update zip file with the new private key
- Performed a DFU update and observed the application running
- Return to bootloader mode through a button press.

☞ Up to this point, the application did not require any modification. The DFU zip file is generated from the existing application hex and generated public key files.

5 Buttonless DFU

If there are no available buttons available to enter bootloader mode through hardware, or to create a more seamless experience for the end user, the process may be started automatically through the "buttonless DFU" feature. This requires modification of the application to add a new service which enables this feature.

A Bluetooth low energy service is a collection of information and behaviors to perform a particular function or feature. In this case, the service will accept a data value (a characteristic) and write it to a register that is persistent across a device reset. The bootloader then reads this register on startup to determine if it should continue with the DFU function, or pass control back to the application. Additional details can be found in this tutorial [17] from Nordic Semiconductor.

In short, this service performs what was originally enabled by pressing button 4 while resetting the Eval board.

The ble_app_buttonless_dfu example will be added to the ble_app_uart example.

To save the work completed so far, copy the modified project a new example folder:

copy C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working_dfu

Change directory into the new folder:

cd C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working_dfu

Open the project file from here:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working_dfu
\pca10056\s140\ses\ble_app_uart_pca10056_s140.emProject
```

5.1 Preprocessor definitions

The project options need modified to use DFU.

Right click on the project and select "Options". Once the options window is open.

☞ Be sure to select the "Common" configuration before editing the values:

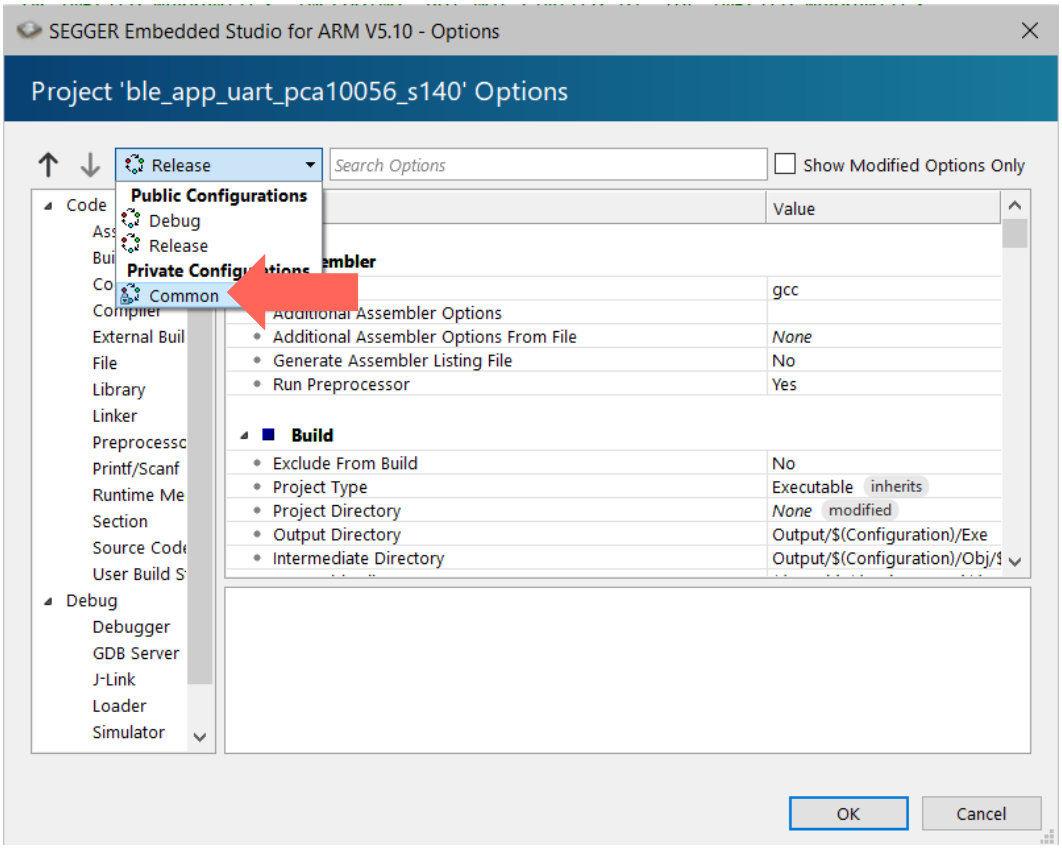


Figure 11: SES project pre-processor definitions

Select the Preprocessor topic under “Code” and add the following to “Preprocessor Definitions”:

```
BL_SETTINGS_ACCESS_ONLY
NRF_DFU_TRANSPORT_BLE=1
```

Add the following to “User Include Directories”:

```
../../../../../../components/libraries/bootloader
../../../../../../components/libraries/bootloader/dfu
../../../../../../components/libraries/bootloader/ble_dfu
```

☞ The last item above may already be in the list. If so, do not add it again.

The final values to adjust are the RAM_START address and RAM_SIZE. These are found in the Linker options, within the Section Placement Macros line.

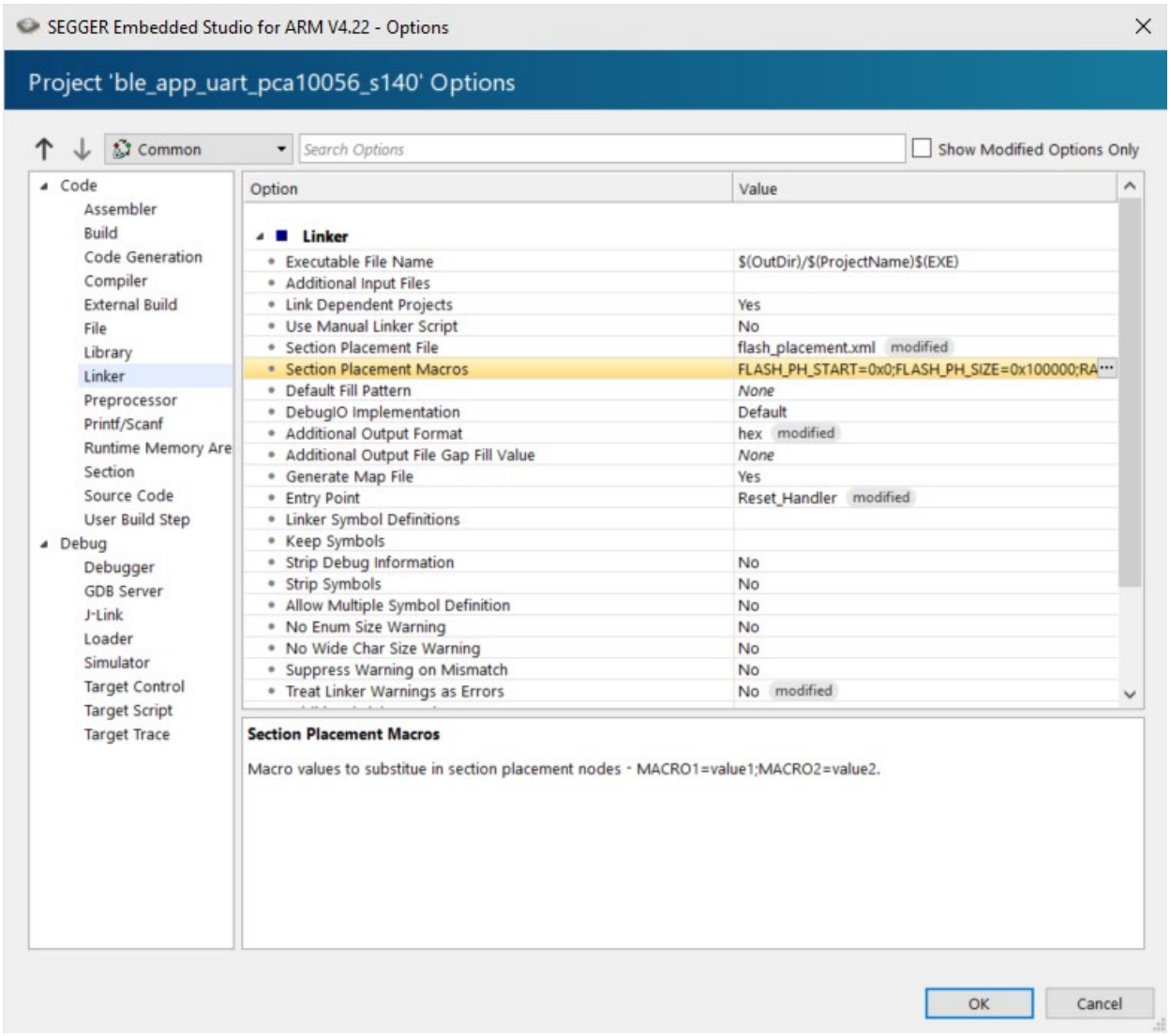


Figure 12: SES project RAM settings

Since the DFU service is being added alongside the Nordic UART Service (NUS), additional memory needs allocated to flash. This is accomplished by increasing the start address of RAM and decreasing RAM size. The new characteristics are 16-bytes each; increase the RAM_START by 0x10 and decrease the RAM_SIZE by 0x10.

For nRF5 SDK v16.0.0, the original values are:

```
RAM_START=0x20002ae8
RAM_SIZE=0x3d518
```

The new values will be:

```
RAM_START=0x20002af8
RAM_SIZE=0x3d508
```

Select “OK” for each open dialog window to return to the main SES window.

⚠ Forgetting to change the RAM start and size is a common error that will cause a hard fault stop.

5.2 sdk_config.h

Open the project `sdk_config.h` file.

There are several settings that need modified to enable DFU. Each line in this list are in separate locations throughout the file. Perform a search for each of these lines and change its value as noted:

```
#define BLE_DFU_ENABLED 1 // was 0
#define NRF_PWR_MGMT_CONFIG_AUTO_SHUTDOWN_RETRY 1 // was 0
#define NRF_SDH_BLE_VS_UUID_COUNT 2 // was 1
#define NRF_SDH_BLE_SERVICE_CHANGED 1 // was 0
```

5.3 Libraries

The DFU library and driver files need added to the project.

Add a new folder in the project called nRF_DFU and add these existing files from the DFU directory of the Bluetooth low energy components:

From the directory:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\components\ble\ble_services\ble_dfu
```

Add these files:

```
ble_dfu.c
ble_dfu_bonded.c
ble_dfu_unbonded.c
```

And from the directory:

```
C:\u-blox\nRF5_SDK_17.0.0_9d13099\components\libraries\bootloader\dfu
```

Add this file:

```
nrf_dfu_svci.c
```

After adding the folder and files, the project will have this section:

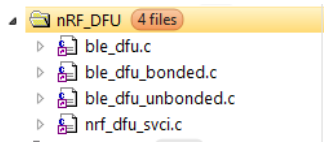


Figure 13: Buttonless UART DFU project - added files

The project still has the same ble_app_uart functionality as the original example application, though now it also includes everything needed to start adding the buttonless DFU function.

5.4 main.c

With the "behind the scenes" items configured, the DFU code can be added to `main.c`.

Several header files are needed. Add these to the other header files near the top of `main.c`:

```
// BEGIN Block Added for DFU

#include "nrf_dfu_ble_svci_bond_sharing.h"
#include "nrf_svci_async_function.h"
#include "nrf_svci_async_handler.h"
#include "ble_dfu.h"
#include "nrf_bootloader_info.h"

// END Block Added for DFU
```

The advertising name will be changed to confirm the DFU version is running. Search for "Nordic_UART" and change it to "UART_DFU":

```
#define DEVICE_NAME "UART_DFU" /**< Name of device. Will be included in the advertising
data. Changed for DFU */
```

A new event handler function is added to act on the data being sent to the DFU service. Place this code above the "assert_nrf_callback" function:

```
/**@brief Function for handling DFU events
 *
 * @details This function is called when entering buttonless DFU
 *
 * @param[in] event Buttonless DFU event.
 */
static void ble_dfu_buttonless_evt_handler(ble_dfu_buttonless_evt_type_t event)
{
    switch (event)
    {
        case BLE_DFU_EVT_BOOTLOADER_ENTER_PREPARE:
            NRF_LOG_INFO("Device is preparing to enter bootloader mode\r\n");
            break;

        case BLE_DFU_EVT_BOOTLOADER_ENTER:
            NRF_LOG_INFO("Device will enter bootloader mode\r\n");
            break;

        case BLE_DFU_EVT_BOOTLOADER_ENTER_FAILED:
            NRF_LOG_ERROR("Device failed to enter bootloader mode\r\n");
            break;
        default:
            NRF_LOG_INFO("Unknown event from ble_dfu.\r\n");
            break;
    }
}
```

The next added function handles the power management. The bootloader service writes a value to a persistent register, then issues a system reset. On restart, the bootloader reads this value to determine whether it should continue, or pass control to the application.

☞ Code can be added here to prevent the DFU from starting if something critical is going on within the application.

Add this function code below the DFU event handler that was just added:

```
/**@brief Function for handling bootloader power management events
 *
 * @details This function is called to set a persistent register which informs the
 *  bootloader it should continue or pass control back to the application
 *
 * @param[in] event Power management event.
 */
static bool app_shutdown_handler(nrf_pwr_mgmt_evt_t event)
{
    switch (event)
    {
        case NRF_PWR_MGMT_EVT_PREPARE_DFU:
            NRF_LOG_INFO("Power management wants to reset to DFU mode\r\n");
            // Change this code to tailor to your reset strategy.
            // Returning false here means that the device is not ready
            // to jump to DFU mode yet.
            //
            // Here is an example using a variable to delay resetting the device:
            //
            /* if (!im_ready_for_reset)
```

```
                {
                    return false;
                }
            */
            break;

        default:
            // Implement any of the other events available
            // from the power management module:
            // -NRF_PWR_MGMT_EVT_PREPARE_SYSOFF
            // -NRF_PWR_MGMT_EVT_PREPARE_WAKEUP
            // -NRF_PWR_MGMT_EVT_PREPARE_RESET
            return true;
    }
    NRF_LOG_INFO("Power management allowed to reset to DFU mode\r\n");
    return true;
}

NRF_PWR_MGMT_HANDLER_REGISTER(app_shutdown_handler, 0); // persistent register for \
                                                           determining DFU status \
                                                           on startup
```

The DFU service is initialized by adding it to the `services_init()` function, **in bold**, italic below:

```
static void services_init(void)
{
    uint32_t err_code;
    ble_nus_init_t nus_init;
    nrf_ble_qwr_init_t qwr_init = {0};

// Initialize Queued Write Module.
    qwr_init.error_handler = nrf_qwr_error_handler;
    err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

// Initialize NUS (Nordic UART Service)
    memset(&nus_init, 0, sizeof(nus_init));
    nus_init.data_handler = nus_data_handler;
    err_code = ble_nus_init(&m_nus, &nus_init);
    APP_ERROR_CHECK(err_code);

// BEGIN Block Added for DFU
// ONLY ADD THIS BLOCK TO THE EXISTING FUNCTION
    // Initialize the DFU service
    ble_dfu_buttonless_init_t dfus_init =
    {
        .evt_handler = ble_dfu_buttonless_evt_handler
    };
    err_code = ble_dfu_buttonless_init(&dfus_init);
    APP_ERROR_CHECK(err_code);
// END Block Added for DFU

}
```

Check that the application compiles without errors.

☞ Do not load the application to the BMD-340-EVAL at this time. A different process will be used to load it to at a later step.

5.5 Hex file generation

Three of the four required components of the project are completed:

- Bootloader
- Application with buttonless DFU code included
- SoftDevice

The last crucial part – bootloader settings – needs generated. The bootloader settings values tell the bootloader about the application:

- Is a valid application present?
- What is the start address of the application?
- What are the version numbers of the bootloader, SoftDevice, and application?

This file is generated by nrfutil with the new application file as input. The output is a new hex file which is then combined with the bootloader, SoftDevice and application hex files. A single hex file is then generated from the four files and used to program a "blank-part" with everything in one pass over the SWD port.

To avoid long file paths when using the nrfutil command, copy the hex files generated up to this point to a common directory:

```
md c:\u-blox\hexfiles
```

Copy the respective hex files to this new location:

```
copy C:\u-blox\nRF5_SDK_17.0.0_9d13099\components\softdevice\s140\hex\s140_nrf52_7.0.1_softdevice.hex
c:\u-blox\hexfiles

copy C:\u-
blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\secure_bootloader\pca10056_s140_ble_working\ses\
Output\Release\Exe\secure_bootloader_ble_s140_pca10056.hex c:\u-blox\hexfiles

copy C:\u-
blox\nRF5_SDK_17.0.0_9d13099\examples\ble_peripheral\ble_app_uart_working_dfu\pca10056\s14
0\ses\Output\Release\Exe\ble_app_uart_pca10056_s140.hex c:\u-blox\hexfiles

cd c:\u-blox\hexfiles
```

Run `nrfutil` to generate the bootloader settings file:

```
nrfutil settings generate --family NRF52840 --application ble_app_uart_pca10056_s140.hex -
-application-version-string "1.2.3" --bootloader-version 0 --bl-settings-version 2
bl_setting.hex

Note: Generating a DFU settings page with backup page included.
This is only required for bootloaders from nRF5 SDK 15.1 and newer.
If you want to skip backup page generation, use --no-backup option.
Generated Bootloader DFU settings .hex file and stored it in: bl_setting.hex
Bootloader DFU Settings:
* File: bl_setting.hex
* Family: NRF52840
* Start Address: 0x000FF000
* CRC: 0x9DA9D5DF
* Settings Version: 0x00000002 (2)
* App Version: 0x000027DB (10203)
* Bootloader Version: 0x00000000 (0)
* Bank Layout: 0x00000000
* Current Bank: 0x00000000
* Application Size: 0x0000BC28 (48168 bytes)
* Application CRC: 0xD5A02BF3
* Bank0 Bank Code: 0x00000001
* Softdevice Size: 0x00000000 (0 bytes)
* Boot Validation CRC: 0xF906A7EC
* SD Boot Validation Type: 0x00000000 (0)
* App Boot Validation Type: 0x00000001 (1)

c:\u-blox\hexfiles>
```

☞ The family of NRF52840 is selected for the BMD-340. See the `--help` output for other nRF5x versions.

All components are now ready:

- Bootloader
- Application with buttonless DFU code included
- SoftDevice
- Bootloader Settings

The Nordic Semiconductor `mergehex` is used to create the resulting hex file. `mergehex` will allow a maximum of three input files, so two steps are required:

```
mergehex --merge bl_setting.hex secure_bootloader_ble_s140_pca10056.hex
s140_nrf52_7.0.1_softdevice.hex --output bl_set_s140.hex

Parsing input hex files.
Merging files.
Storing merged file.

mergehex --merge bl_set_s140.hex ble_app_uart_pca10056_s140.hex --output
bl_set_s140_app.hex

Parsing input hex files.
Merging files.
Storing merged file.
```

This final hex file – `bl_set_s140_app.hex` – with all the parts included is what is typically used on a production line, or for pre-programming the module ahead of assembly into the end-product.

☞ Other details may be added to the hex file, including the public Bluetooth address noted in section 2.1 above.

6 Test DFU

6.1 Program and test application

Everything is now in place to test the new application.

Erase the BMD-340 to ensure nothing is left-over from previous work:

```
nrfjprog -f nrf52 --recover
Recovering device. This operation might take 30s.
Erasing user code and UICR flash areas.
```

The final step is to load the hex file to the BMD-340:

```
nrfjprog --program bl_set_s140_app.hex
Parsing hex file.
Reading flash area to program to guarantee it is erased.
Checking that the area to write is not protected.
Programming device.
```

Power-cycle the BMD-340-EVAL. The application will start advertising and blink LED1 as before.

Connect to it with nRF Connect:

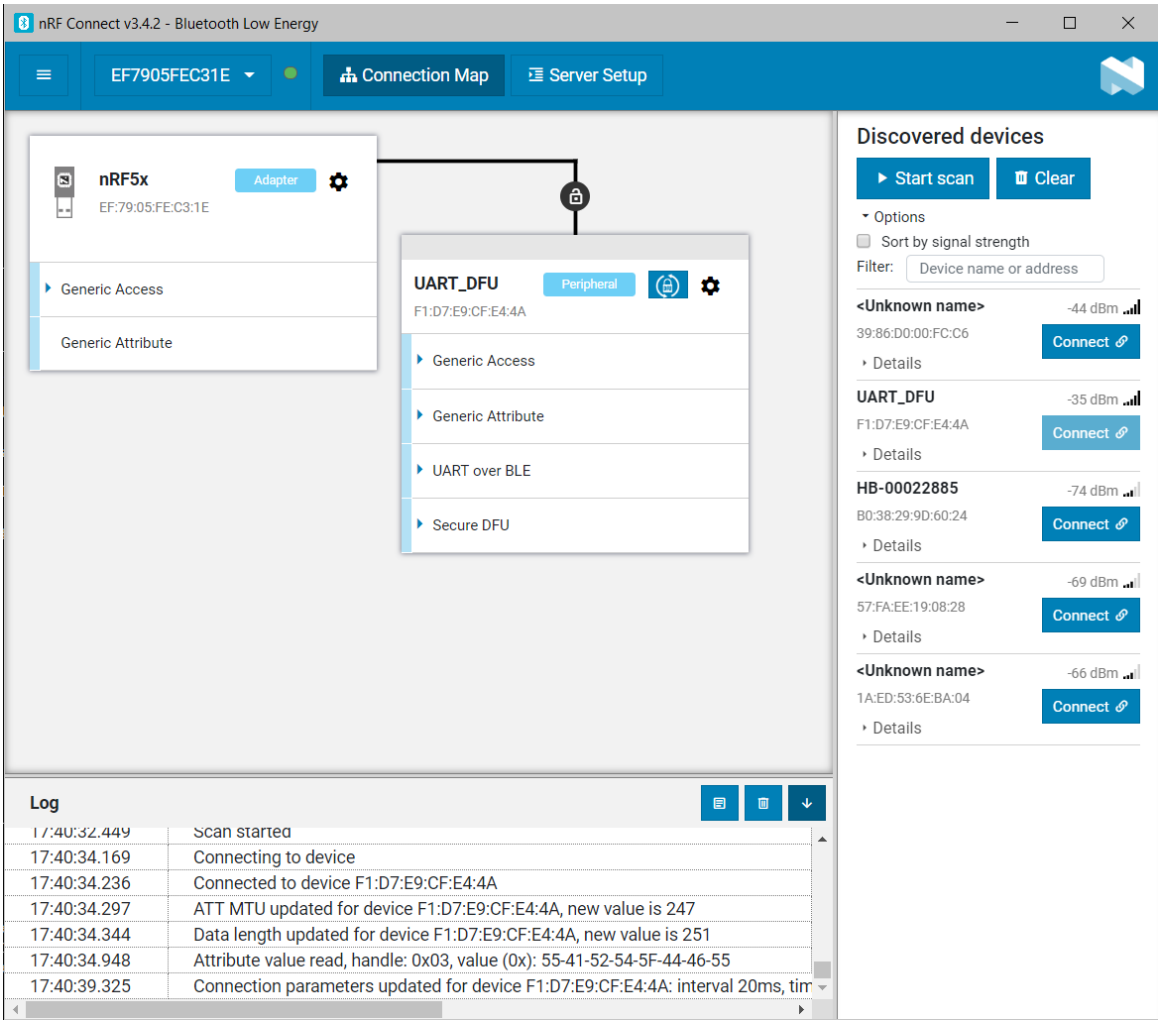


Figure 14: nRF Connect with application showing secure DFU icon

The bootloader icon is shown as with the button DFU example. The new name, UART_DFU, and the Nordic UART Service, UART over BLE, are also present.

6.2 Prepare and test update

Since the application is already compiled to a hex file, it can be used as the source application file. The application version is stepped to inform the bootloader that it is a new, valid update is available.

In the hexfile directory, `nrfutil` is used to generate the new zip file:

```
nrfutil pkg generate --application ble_app_uart_pca10056_s140.hex --application-version-string "2.0.0" --hw-version 52 --sd-req 0xCA --key-file C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\dfu_private_key.pem app_v2.zip
```

From nRF Util, start the DFU process and send the update:

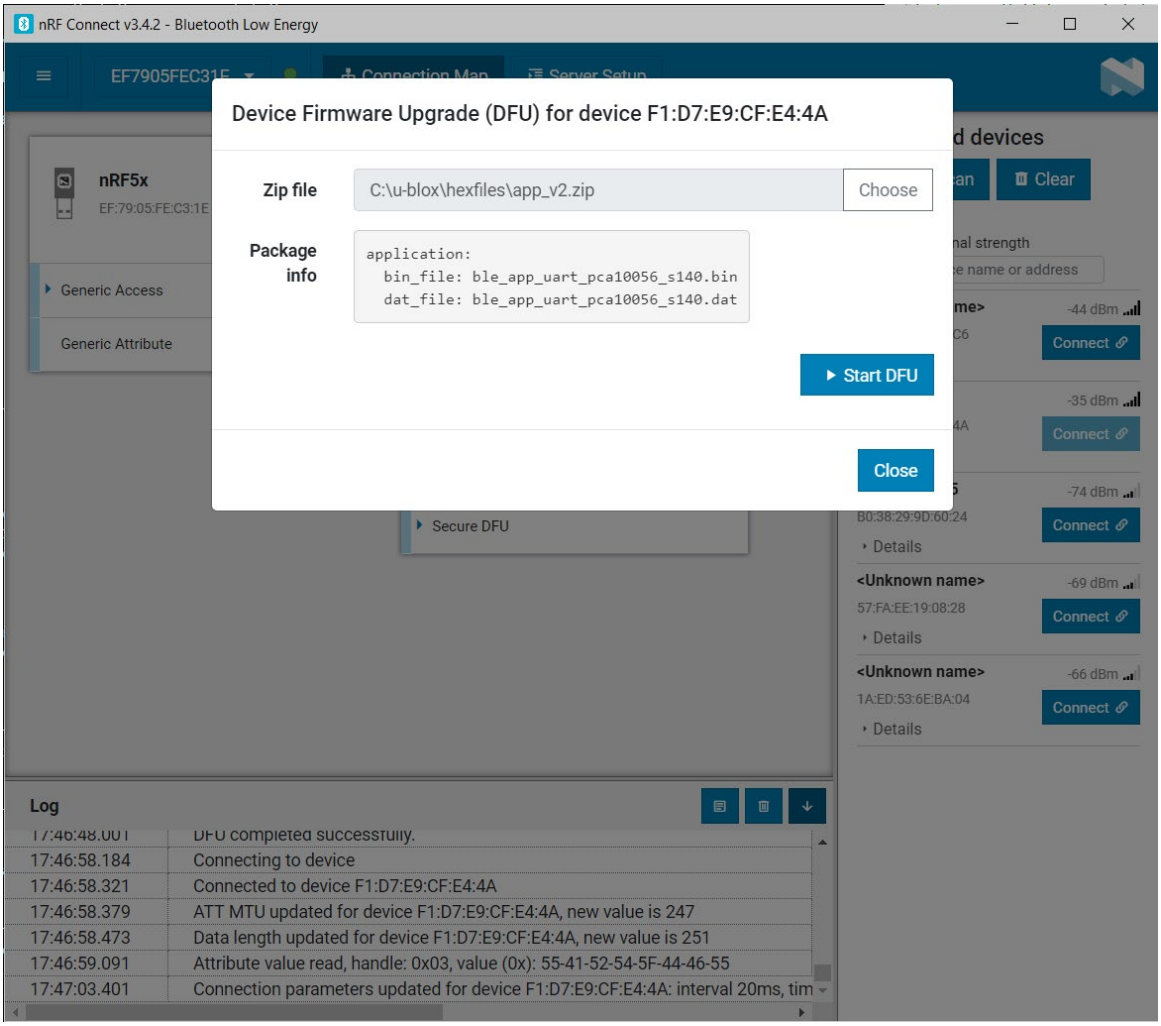


Figure 15: SES ready for update from application

To observe version checking, create another zip file, with a version string of “1.0.0”.

```
nrfutil pkg generate --application ble_app_uart_pca10056_s140.hex --application-version-string "1.0.0" --hw-version 52 --sd-req 0xCA --key-file C:\u-blox\nRF5_SDK_17.0.0_9d13099\examples\dfu\dfu_private_key.pem app_v1.zip
```

If the new file is selected with nRF Connect, an error will be generated since a newer version is present on the BMD-340.

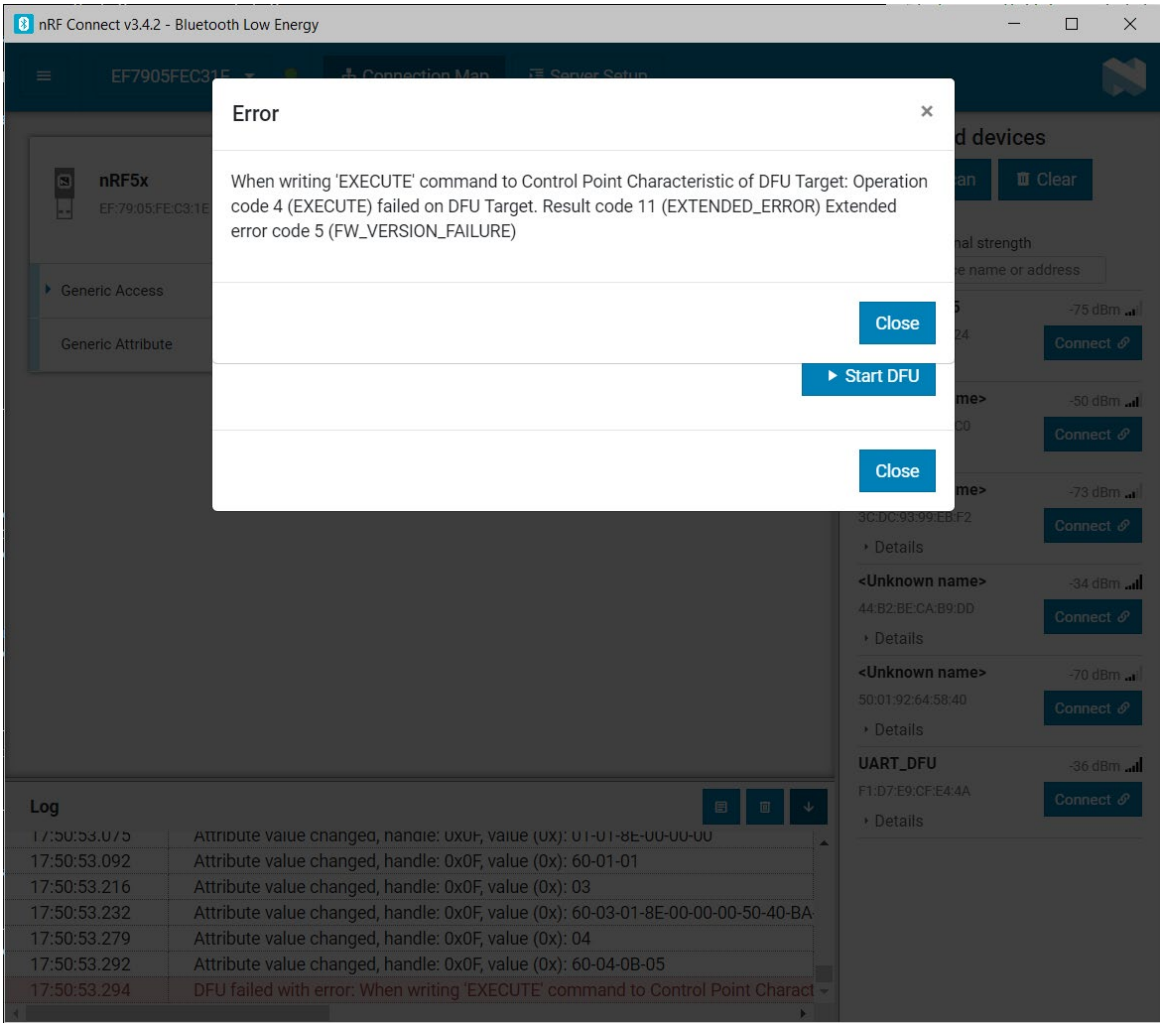


Figure 16: nRF Connect showing error when attempting an "old" update

Version checking is performed for all three components: bootloader, SoftDevice, and application.

There are many more aspects of the Nordic DFU service, such as activating additional transports (UART, USB, Bluetooth Mesh, Zigbee, and Thread), additional security with bond forwarding, and so on. For further information, see reference [18].

Appendix

A Glossary

Abbreviation	Definition
ASCII	American Standard Code for Information Interchange
ARM	Arm (Advanced RISC Machines) Holdings
CPU	Central Processing Unit
DFU	Device Firmware Update
DK	Development Kit (see EVK)
EVK	EValuation Kit
FICR	Factory Information Control Registers
GCC	GNU Compiler Collection
GNU	Recursive acronym “GNU Not Unix”
IDE	Integrated Development Environment
LED	Light Emitting Diode
MAC address	Media Access Control address: a unique identifier assigned to a network interface controller
NUS	Nordic UART Service
RAM	Random Access Memory
SDK	Software Development Kit
SES	SEGGER Embedded Studio
SoftDevice	Bluetooth low energy stack provided by Nordic Semiconductor
TLA	Three Letter Acronym
UART	Universal Asynchronous Receiver Transmitter
UICR	User Information Control Registers
USB	Universal Serial Bus
WSL	Windows Subsystem for Linux

Table 2: Explanation of the abbreviations and terms used

Related documents

[1] ANNA-B112 system integration manual, UBX-18009821
[2] NINA-B1 system integration manual, UBX-15026175
[3] NINA-B3 system integration manual, UBX-17056748
[4] BMD-345 data sheet, UBX-19039908
[5] u-blox package information guide, UBX-14001652
[6] Using the public IEEE address from UICR application note, UBX-19050198
[7] Nordic Semiconductor nRF5 SDK
[8] Nordic Semiconductor nRF Connect for Desktop
[9] Nordic Semiconductor nRF Command Line Tools
[10] Nordic Semiconductor nRF Util GitHub repository
[11] Python scripting language download
[12] Micro_ecc encryption libraries backend instructions
[13] Git version control download
[14] GNU Arm Embedded Toolchain (GCC for Arm) download
[15] GNU make download
[16] UART example test instructions
[17] Nordic Semiconductor bootloader details
[18] Nordic Semiconductor Bluetooth services tutorial
[19] Nordic Semiconductor Bootloader and DFU modules

☞ For product change notifications and regular updates of u-blox documentation, register on our website, www.u-blox.com.

Revision history

Revision	Date	Name	Comments
R01	16-Dec-2019	brec	Initial release
R02	02-May-2020	brec	Added reference to UBX-19050198, corrected minor typographical errors
R03	08-Jan-2021	brec	Changed style, updated for nRF5 SDK v17.0.0, nRF Connect v3.4.2

Contact

For complete contact information, visit us at www.u-blox.com.

u-blox Offices

North, Central and South America

u-blox America, Inc.

Phone: +1 703 483 3180
E-mail: info_us@u-blox.com

Regional Office West Coast:

Phone: +1 408 573 3640
E-mail: info_us@u-blox.com

Technical Support:

Phone: +1 703 483 3185
E-mail: support@u-blox.com

Headquarters
Europe, Middle East, Africa

u-blox AG

Phone: +41 44 722 74 44
E-mail: info@u-blox.com
Support: support@u-blox.com

Asia, Australia, Pacific

u-blox Singapore Pte. Ltd.

Phone: +65 6734 3811
E-mail: info_ap@u-blox.com
Support: support_ap@u-blox.com

Regional Office Australia:

Phone: +61 2 8448 2016
E-mail: info_anz@u-blox.com
Support: support_ap@u-blox.com

Regional Office China (Beijing):

Phone: +86 10 68 133 545
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Chongqing):

Phone: +86 23 6815 1588
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shanghai):

Phone: +86 21 6090 4832
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shenzhen):

Phone: +86 755 8627 1083
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office India:

Phone: +91 80 405 092 00
E-mail: info_in@u-blox.com
Support: support_in@u-blox.com

Regional Office Japan (Osaka):

Phone: +81 6 6941 3660
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Japan (Tokyo):

Phone: +81 3 5775 3850
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Korea:

Phone: +82 2 542 0861
E-mail: info_kr@u-blox.com
Support: support_kr@u-blox.com

Regional Office Taiwan:

Phone: +886 2 2657 1090
E-mail: info_tw@u-blox.com
Support: support_tw@u-blox.com