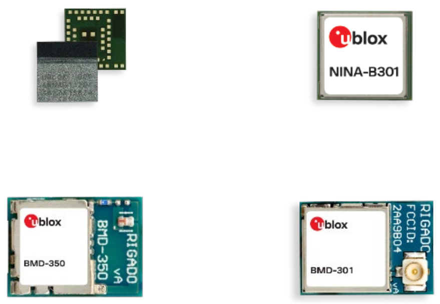
Add Nordic Semiconductor DFU to SDK Example

Bluetooth low energy

Application note



Abstract

We'll take an existing Bluetooth low energy peripheral example and add Nordic Semiconductor's "buttonless" DFU features to the application, then proceed to create an update zip file and use it to update the firmware on a Nordic Semiconductor based module.

www.u-blox.com
UBX-19050198 - R01

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	R01	16-Dec-2019
Disclosure restriction		

This document applies to the following products:

Product name
ANNA-B112
BMD-300
BMD-301
BMD-340
BMD-341
BMD-345
BMD-350
BMD-380
NINA-B111
NINA-B112
NINA-B301
NINA-B302
NINA-B306

Contents

1 Software preparation

1.1 Nordic Semiconductor nRF5 SDK

Let's start with a fresh unzip of the SDK. This gives us a known starting point. Nordic's nRF5 SDK many be found here: https://www.nordicsemi.com/Software-and-Tools/Software/nRF5-SDK

For this article, we'll place the SDK in the following folder:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2
```

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 of u-blox's 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

We'll use several utilities from Nordic Semiconductor throughout the article. Let's install them now:

1.3.1 nRF Connect

https://www.nordicsemi.com/Software-and-Tools/Development-Tools/nRF-Connect-for-desktop/Download#infotabs

1.3.2 nRF command line tools (nrfjprog)

https://www.nordicsemi.com/Software-and-Tools/Development-Tools/nRF5-Command-Line-Tools/Download#infotabs

1.3.3 nRF Util

https://www.nordicsemi.com/Software-and-Tools/Development-Tools/nRF-Util

https://github.com/NordicSemiconductor/pc-nrfutil

nRF Util depends on Python: https://www.python.org/downloads/ The current release (v5.2.0) requires Python 2.x. Newer releases will move to Python v3.x.

☞ nRF Util can also be downloaded as a pre-compiled EXE for Windows that is not dependent on Python also being installed. If this EXE is used, be sure to add its location to your PATH environment variables.

⚠ nRF Util v6.0.0a0 and v6.0.0a1 have a bug in the public key generation that will cause the bytes to be reversed resulting in an "INVALID_OBJECT" error. Use nRF Util v5.2.0 until this issue is fixed.

1.4 Encryption libraries

Next, let's add the encryption libraries that are required for the signed update files we'll create for use with DFU. There are a couple preliminary steps as outlined in Nordic's instructions here:

https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/lib_crypto_backend_micro_ecc.html

1.4.1 Install git

git is a useful tool to have available. macOS and Linux already support it with X-Code or right out of the box. For Windows, install it from https://git-scm.com/. Any other form of git may be used, such as with 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 Install GCC for ARM

The GCC for ARM compiler is used to compile the micro-ecc libraries. Download the installer here:

https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads

The nRF5 SDK v16 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 running through the installer, be sure to select the option to "Add path to environment variable".

1.4.3 Install GNU make

We need the "make" utility to manage the build process. There are several ways to install it. As with make, for macOS and Linux it's part of the usual development tools. For Windows, we'll use the installer here: http://gnuwin32.sourceforge.net/packages/make.htm

We need to 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, we can use a shortcut provided within the SDK to do the rest of the work for us. Open a Command window and navigate to the directory

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\external\micro-ecc
```

There you will find a batch file (and a script for macOS and Linux) called "build_all". Go ahead and run that to download micro-ecc and build.

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\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_16.0.0_98a08e2/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_16.0.0_98a08e2/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_16.0.0_98a08e2/external/micro-
ecc/nrf52nf_keil/armgcc'

C:\u-blox\nRF5_SDK_16.0.0_98a08e2\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 article we'll use BMD-340-EVAL boards and a nRF52840 USB dongle.

You can also use any of the nRF52840 or nRF52832 boards: 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 alternate board is being used, be sure to adjust the directories for your 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.

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, you cannot update the SoftDevice (Bluetooth Stack).

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.

In order to save this address for later, use nrfjprog to read the User Information Configuration Register (UICR) area. We'll use the file name evk_uicr.hex. You may name this as you wish.

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

Save the hex file in a convenient location.

If you should need to restore the UICR, simply program it with:

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

The example we use 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). We have another application note that describes how to add code to read and assign the public Bluetooth address held in the UICR.

3 Application firmware

Now that everything is ready, let's go into the example application called ble_app_uart:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart>
```

So we don't modify the original, we'll make a copy of the project and all its subdirectories and only modify the files in the working directory:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working>
```

For this article, we'll use the BMD-340-EVAL board, which is functionally equivalent to the nRF52840 DK. Throughout the SDK, this board is called by its board number, "pca10056".

Navigate to the SES folder and open the project:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working\pca10056\s1
40\ses\ble_app_uart_pca10056_s140.emProject
```

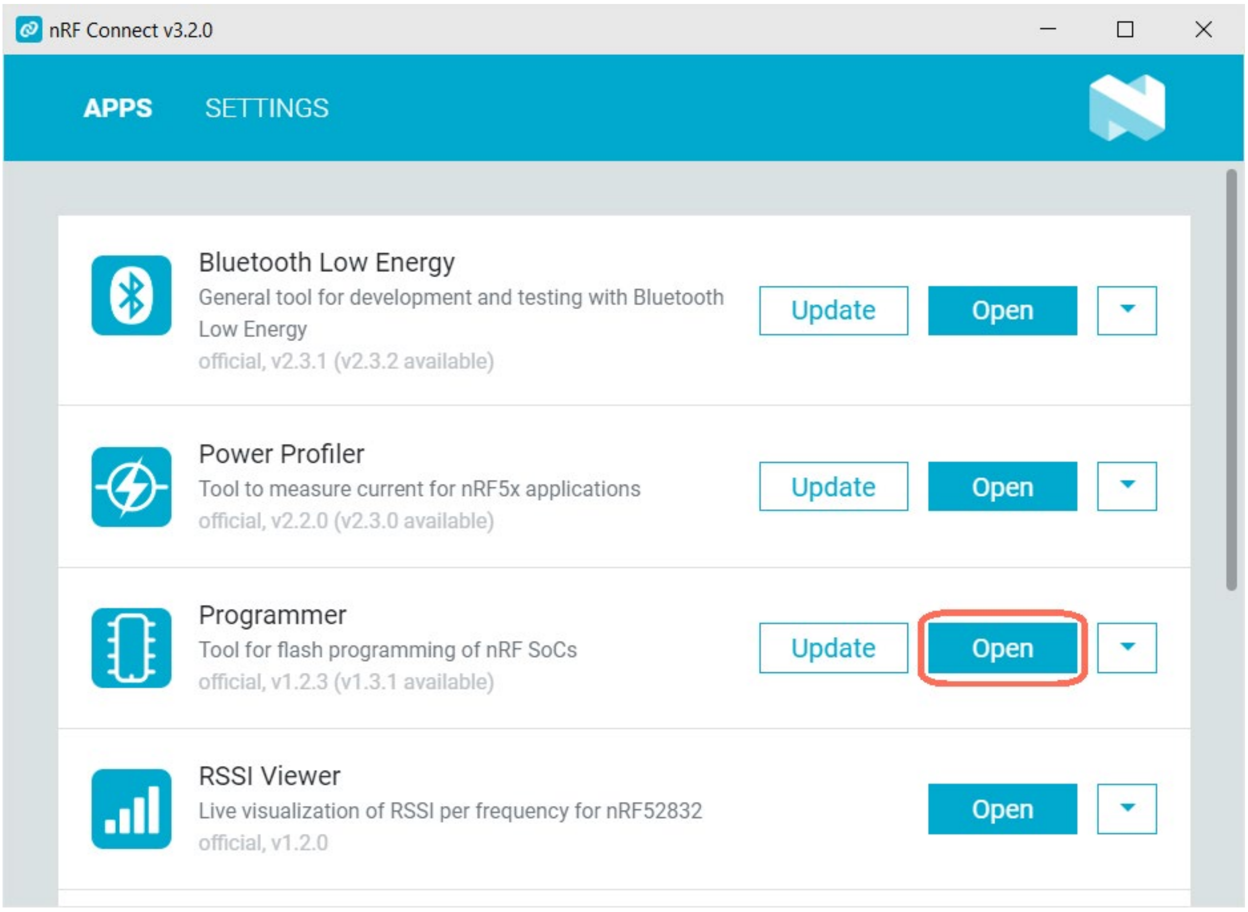
Let's compile and load the application to the BMD-340-EVAL, just to be sure it's working before we start fiddling with it. Details for testing the functionality are located here:

https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/ble_sdk_app_nus_eval.html

Have a quick look at the Hex file to see the memory layout. It's located in the Output directory in the example:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working\pca10056\s1
40\ses\Output\Release\Exe\ble_app_uart_pca10056_s140.hex
```

You can open nRF Connect Programmer then add the hex file, or just drag it into the "File Memory Layout" section:



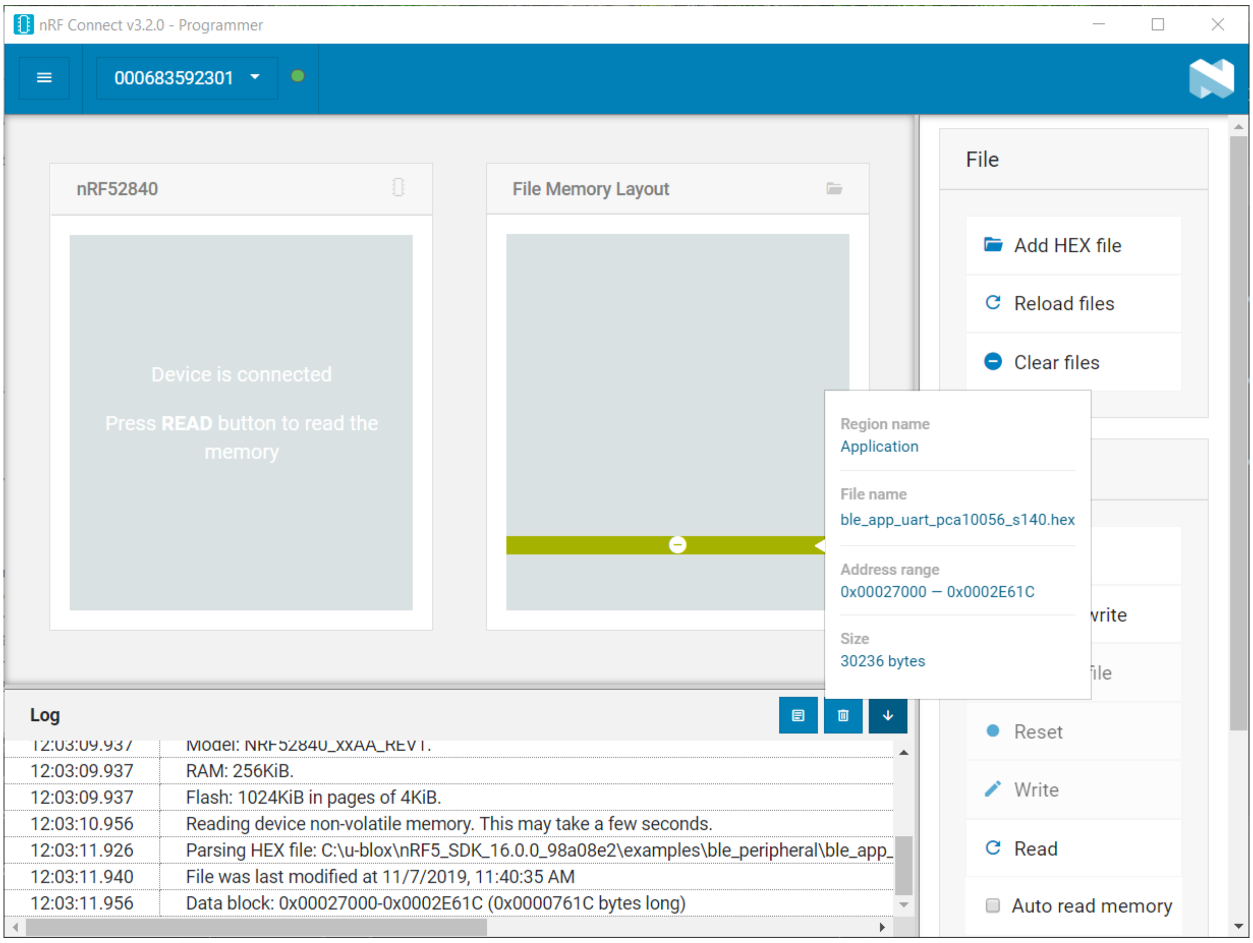
Bluetooth Low Energy
General tool for development and testing with Bluetooth Low Energy
official, v2.3.1 (v2.3.2 available)

Power Profiler
Tool to measure current for nRF5x applications
official, v2.2.0 (v2.3.0 available)

Programmer
Tool for flash programming of nRF SoCs
official, v1.2.3 (v1.3.1 available)

RSSI Viewer
Live visualization of RSSI per frequency for nRF52832
official, v1.2.0

Figure 1: nRF Connect main window (programmer)



Region name
Application
File name
ble_app_uart_pca10056_s140.hex
Address range
0x00027000 — 0x0002E61C
Size
30236 bytes

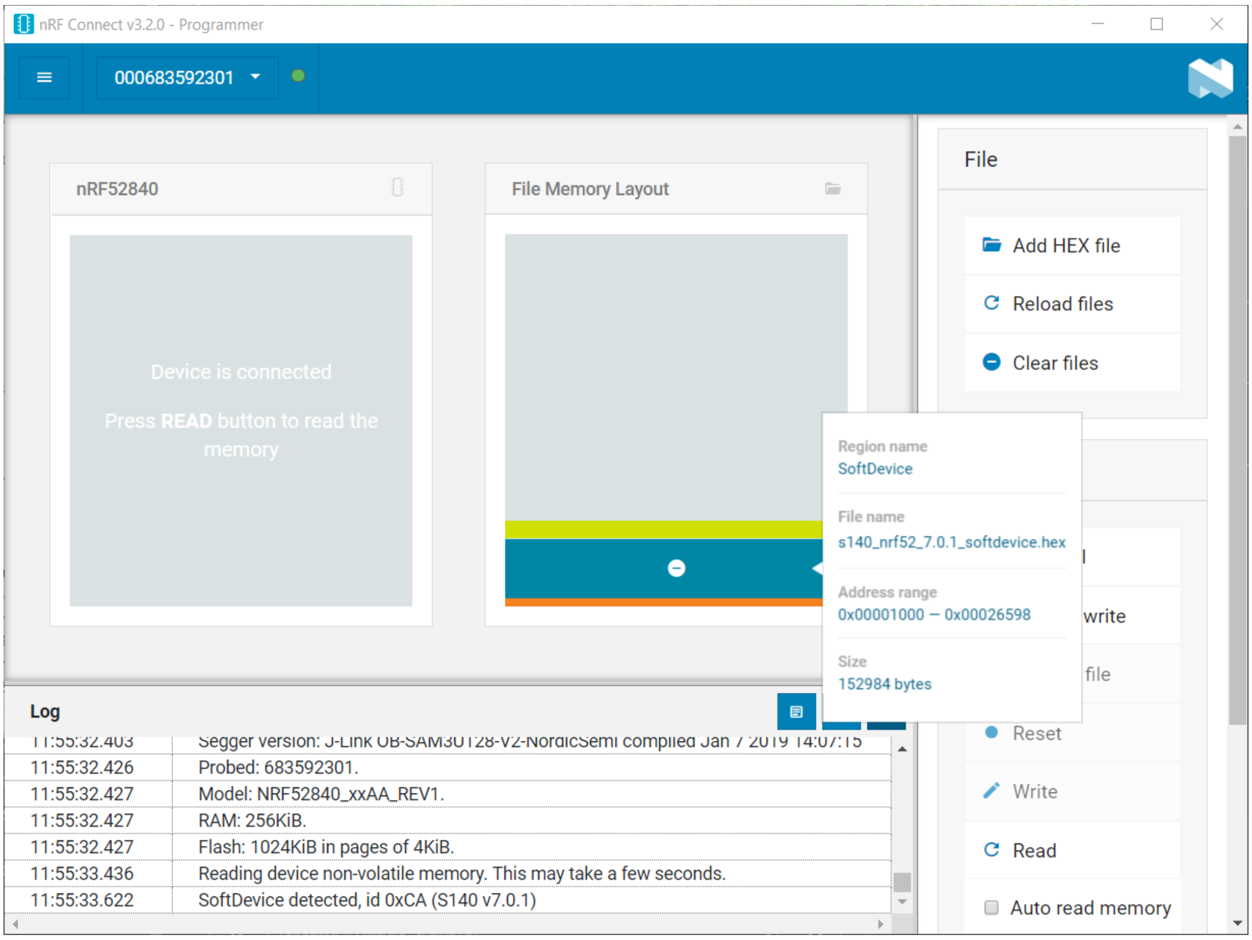
```
12:03:09.937   Model: NRF52840_XXAA_REV1.
12:03:09.937   RAM: 256KiB.
12:03:09.937   Flash: 1024KiB in pages of 4KiB.
12:03:10.956   Reading device non-volatile memory. This may take a few seconds.
12:03:11.926   Parsing HEX file: C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_
12:03:11.940   File was last modified at 11/7/2019, 11:40:35 AM
12:03:11.956   Data block: 0x00027000-0x0002E61C (0x0000761C bytes long)
```

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_16.0.0_98a08e2\components\softdevice\s140\hex\s140_nrf52_7.0.1_softdevi
ce.hex
```

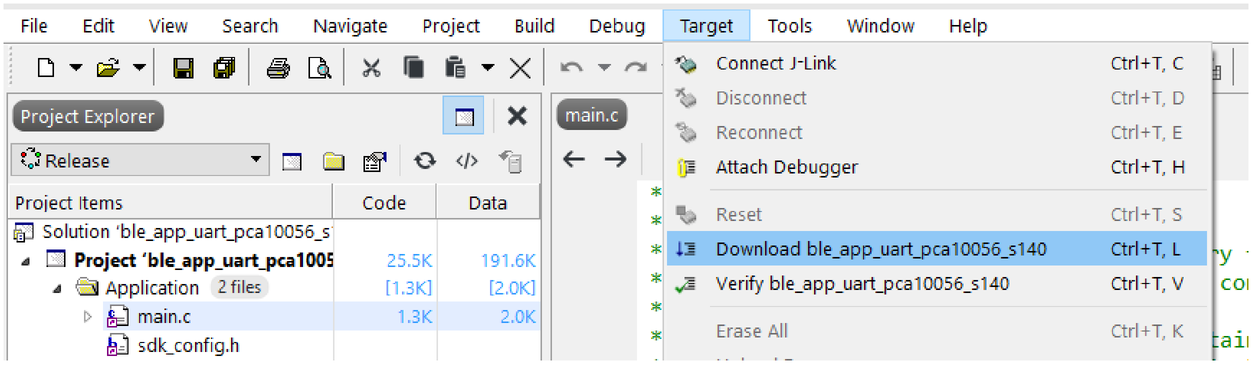
Now both are shown, along with the MBR:



```
11:55:32.403   Segger version: J-Link OB-SAM3U128-V2-NordicSemi compiled Jan 7 2019 14:07:15
11:55:32.426   Probed: 683592301.
11:55:32.427   Model: NRF52840_xxAA_REV1.
11:55:32.427   RAM: 256KiB.
11:55:32.427   Flash: 1024KiB in pages of 4KiB.
11:55:33.436   Reading device non-volatile memory. This may take a few seconds.
11:55:33.622   SoftDevice detected, id 0xCA (S140 v7.0.1)
```

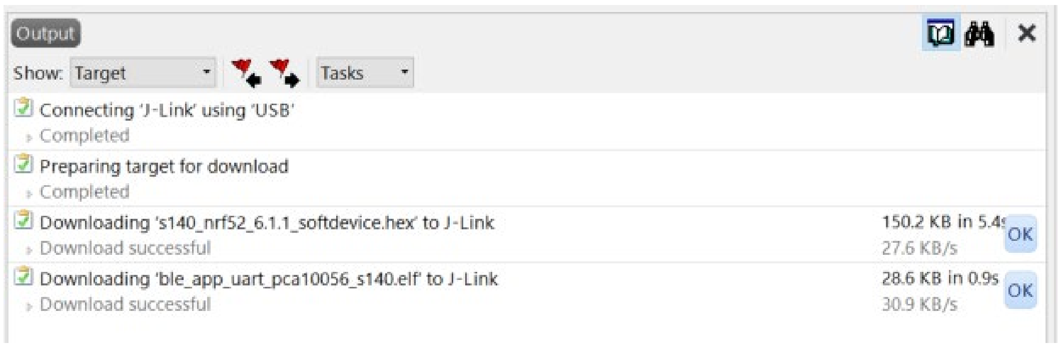
Figure 3: nRF Connect showing application with SoftDevice

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



Project Items	Code	Data
Solution 'ble_app_uart_pca10056_s		
Project 'ble_app_uart_pca1005	25.5K	191.6K
Application 2 files	[1.3K]	[2.0K]
main.c	1.3K	2.0K
sdk_config.h		

Figure 4: SES download application to target



```
Connecting 'J-Link' using 'USB'
  Completed
Preparing target for download
  Completed
Downloading 's140_nrf52_6.1.1_softdevice.hex' to J-Link          150.2 KB in 5.4s   OK
  Download successful                                            27.6 KB/s
Downloading 'ble_app_uart_pca10056_s140.elf' to J-Link           28.6 KB in 0.9s    OK
  Download successful                                            30.9 KB/s
```

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

Now that the application runs correctly, we can work with the bootloader. Nordic Semiconductor provides several DFU examples. The Secure Bootloader example is the one we'll use. See the Nordic Semiconductor InfoCenter for details about this secure bootloader:

https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/ble_sdk_app_dfu_bootloader.html

On your local drive, navigate to the secure_bootloader DFU directory of the examples.

As we did with the UART application, we'll make a copy of the DFU project and its subdirectories. Copy the directory:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\dfu\secure_bootloader\pca10056_s140_ble
```

to

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\dfu\secure_bootloader\pca10056_s140_ble_working
```

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

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\dfu\secure_bootloader\pca10056_s140_ble_working
\ses\secure_bootloader_ble_s140_pca10056.emProject
```

If we just try to compile the bootloader as-is, we'll get 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 supplies an example key along with some test files. These are only made available so we can get used to working with updating a file on a device; however, they cannot be used for production.

Let's do as we're instructed - generate our own keys.

4.2 Key generation

We'll use nRF Util to generate our own set of keys. Let's 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
```

Now we can generate our own keys. First we 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
```

Of course, your public and private keys will have different values.

Now go back to the project in SES and compile the DFU bootloader. There should not be any errors.

Before we program the Eval board, let's erase it fully:

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

Go into the SES project and load the DFU and SoftDevice the same way we did for the UART project.

Both LED1 and LED2 should light up solid. This indicates the BMD-340 is in bootloader mode.

4.3 Firmware update package generation

We'll set the Eval board aside for a moment while we generate and sign the update zip file with the private key we just generated.

Go back into the output directory of the ble_app_uart example that we compiled earlier.

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working\pca10056\s1
40\ses\Output\Release\Exe
```

We'll break out nrfutil again, but this time to generate the zip file:

Running nrfutil with the --help option shows quite a number of available options.

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

For this example, we'll use the following options:

Option	Description
`--application ble_app_uart_pca10056_s140.hex`	The compiler output hex file
`--application-version-string "1.0.0"`	Text version string This text version string is not checked. Only the "application-version" above is checked to be sure you're programming the same or newer application. ☞ The version number is up to your versioning practices - here's one possibility: https://semver.org/
`--hw-version 52`	The default number "52" is used since we are working with a nRF52xxx device. This can actually be any integer as well, for example to reflect the host board version. If you cannot load new code on an old 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.
`--key-file c:\u-blox\nRF5_SDK_16.0.0_98a08e2 \examples\dfu\dfu_private_key.pem`	The path and filename of the private .pem key file that we just generated.
`app_v1.zip`	The output filename

Table 1: nrfutil package generation options

☞ As noted above, v5.2.0 of nrfutil should be used for the time being. The following SoftDevice v7.x.x codes are not listed in the help output, but are able to be used with nrfutil v5.2.0:

```
|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|
```

Putting it all together:

```
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_16.0.0_98a08e2\examples\dfu\dfu_private_key.pem app_v1.zip
Zip created at app_v1.zip
```

Let's give the new file a try.

Connect a second EVK or nRF52840 Dongle to your computer. Open nRF Connect and select the "Bluetooth Low Energy" option:

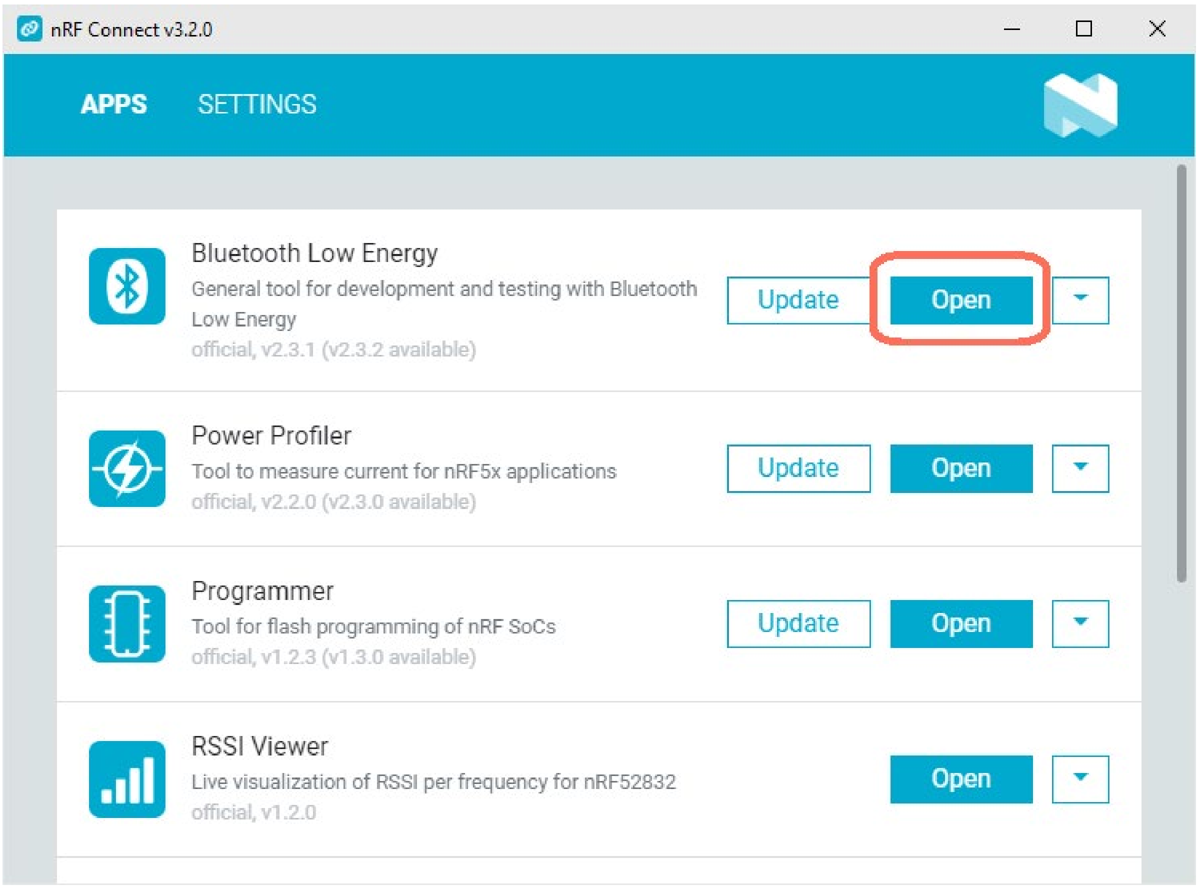
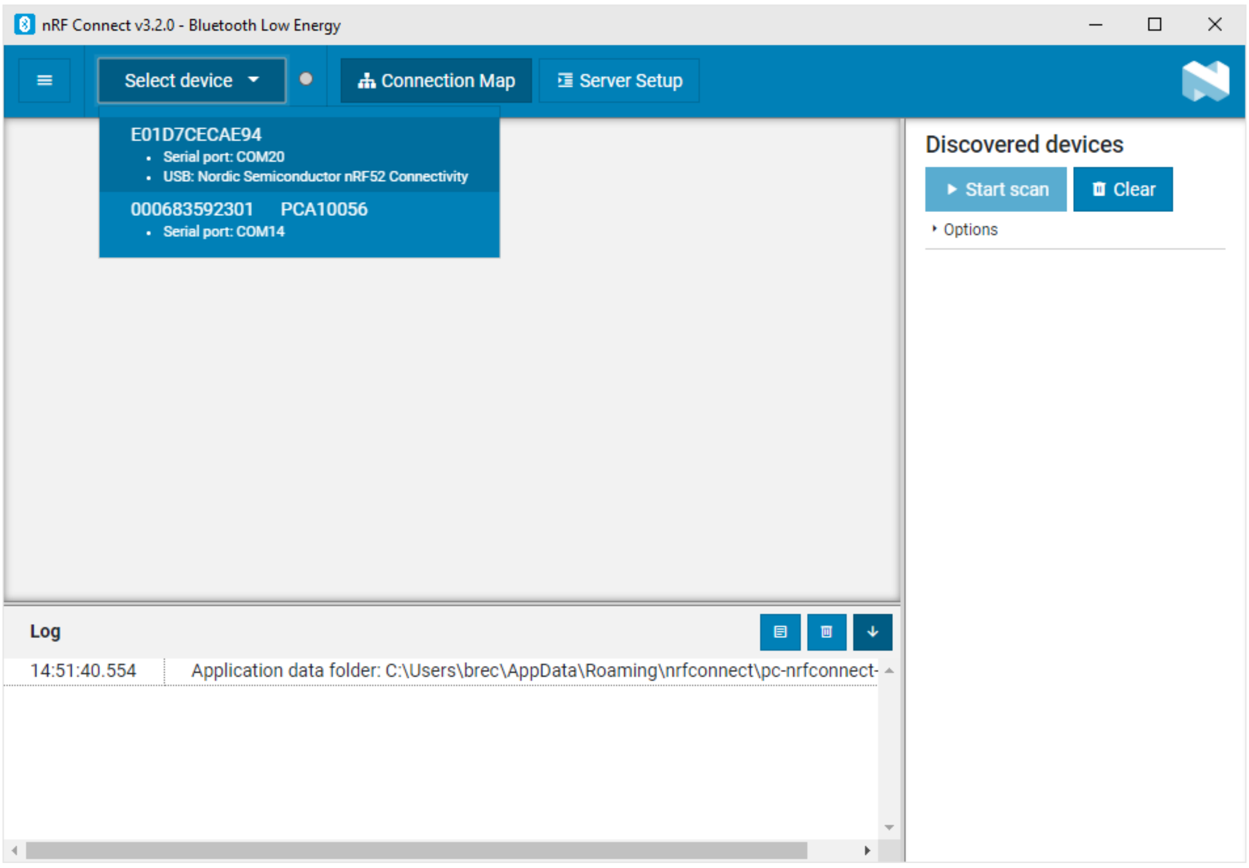

nRF Connect v3.2.0
APPS
SETTINGS
Bluetooth Low Energy
General tool for development and testing with Bluetooth Low Energy
official, v2.3.1 (v2.3.2 available)
Update
Open
Power Profiler
Tool to measure current for nRF5x applications
official, v2.2.0 (v2.3.0 available)
Update
Open
Programmer
Tool for flash programming of nRF SoCs
official, v1.2.3 (v1.3.0 available)
Update
Open
RSSI Viewer
Live visualization of RSSI per frequency for nRF52832
official, v1.2.0
Open

Figure 6: nRF Connect main window (Bluetooth low energy)

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

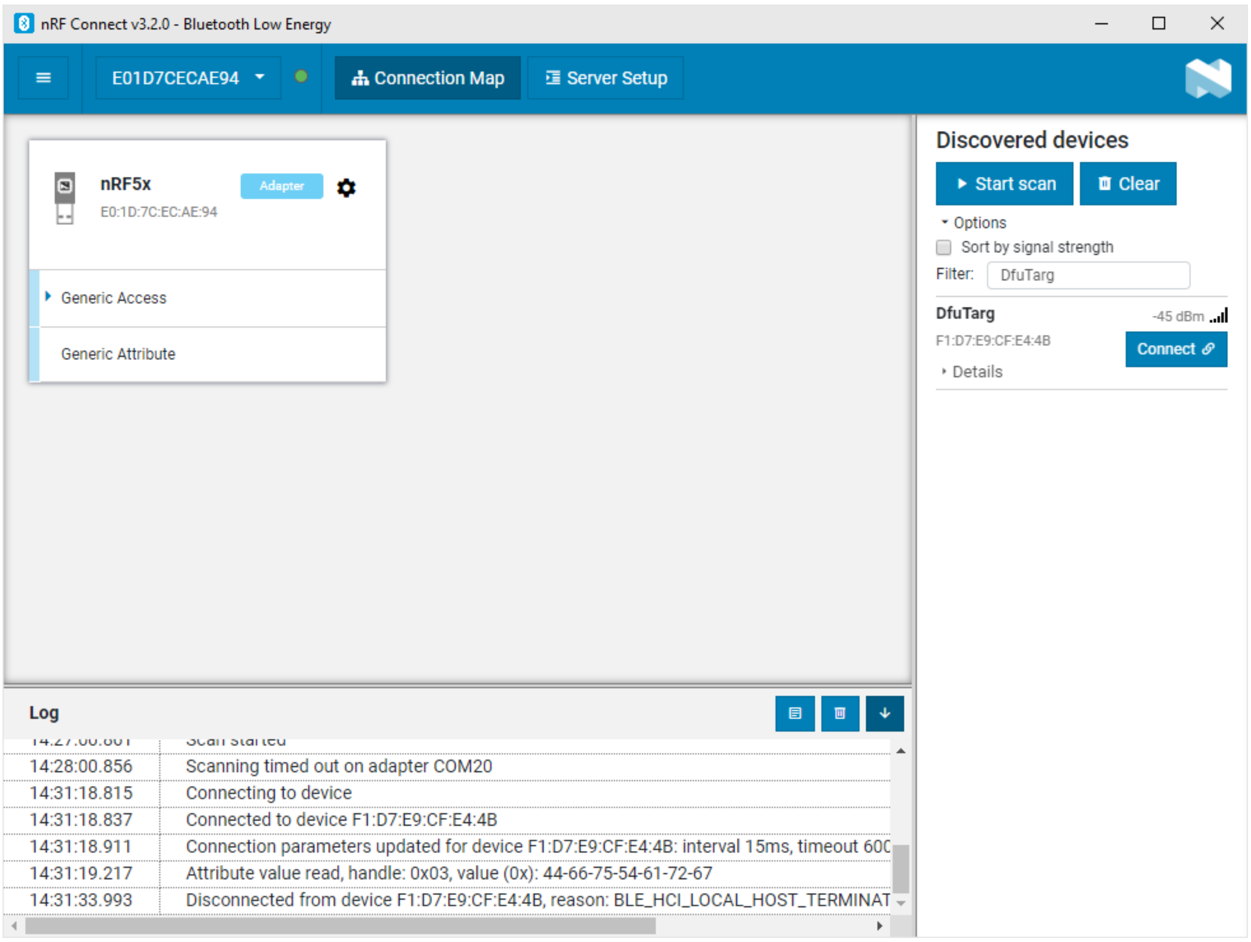
☞ If you see a notice to program the connectivity firmware, select yes.



```
14:51:40.554    Application data folder: C:\Users\brec\AppData\Roaming\nrfconnect\pc-nrfconnect-
```

Figure 7: nRF Connect Bluetooth Low Energy window

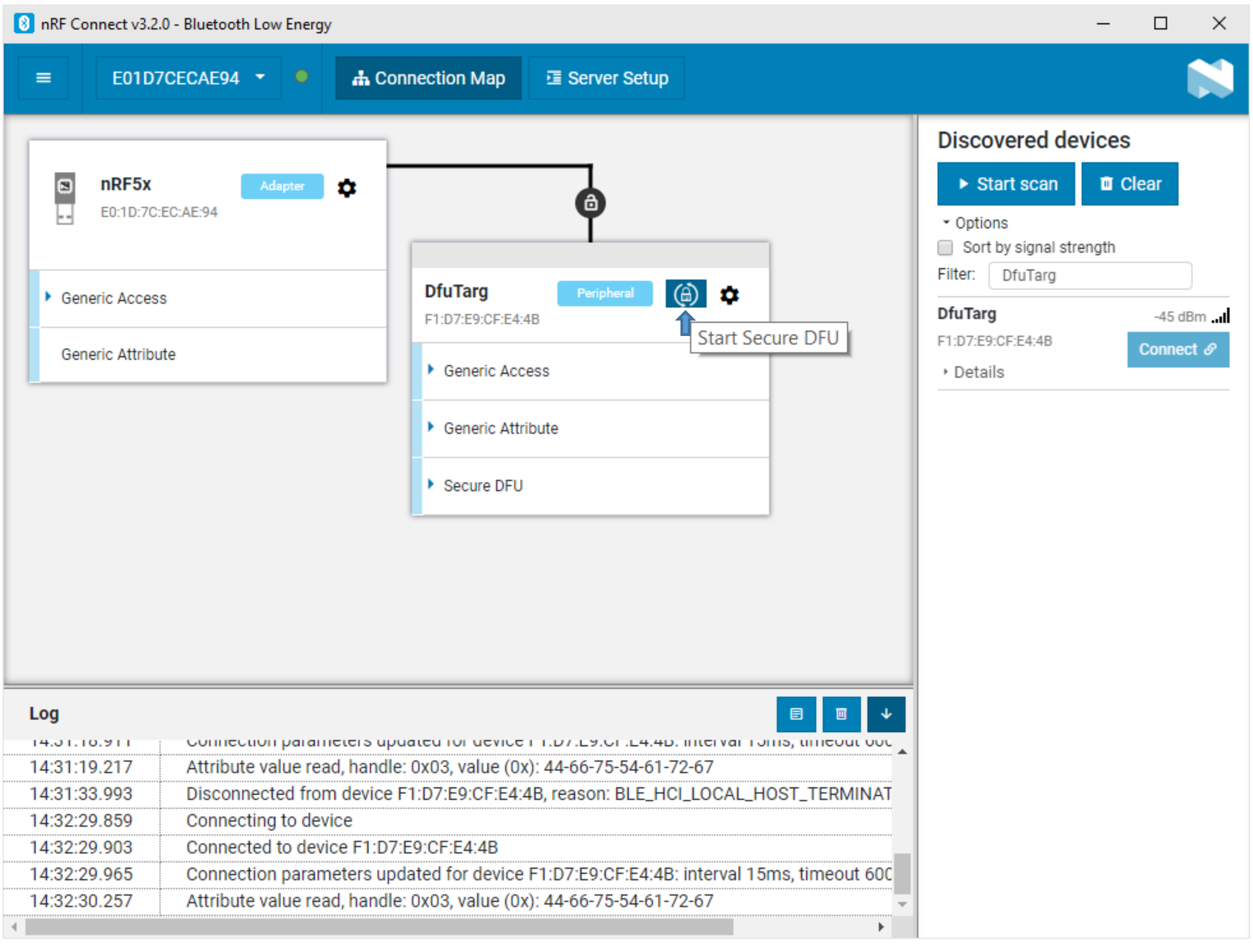
Start the scan and look for the Eval board advertising "DfuTarg":



```
14:28:00.856    Scanning timed out on adapter COM20
14:31:18.815    Connecting to device
14:31:18.837    Connected to device F1:D7:E9:CF:E4:4B
14:31:18.911    Connection parameters updated for device F1:D7:E9:CF:E4:4B: interval 15ms, timeout 600
14:31:19.217    Attribute value read, handle: 0x03, value (0x): 44-66-75-54-61-72-67
14:31:33.993    Disconnected from device F1:D7:E9:CF:E4:4B, reason: BLE_HCI_LOCAL_HOST_TERMINAT
```

Figure 8: nRF Connect showing scanned bootloader DfuTarg

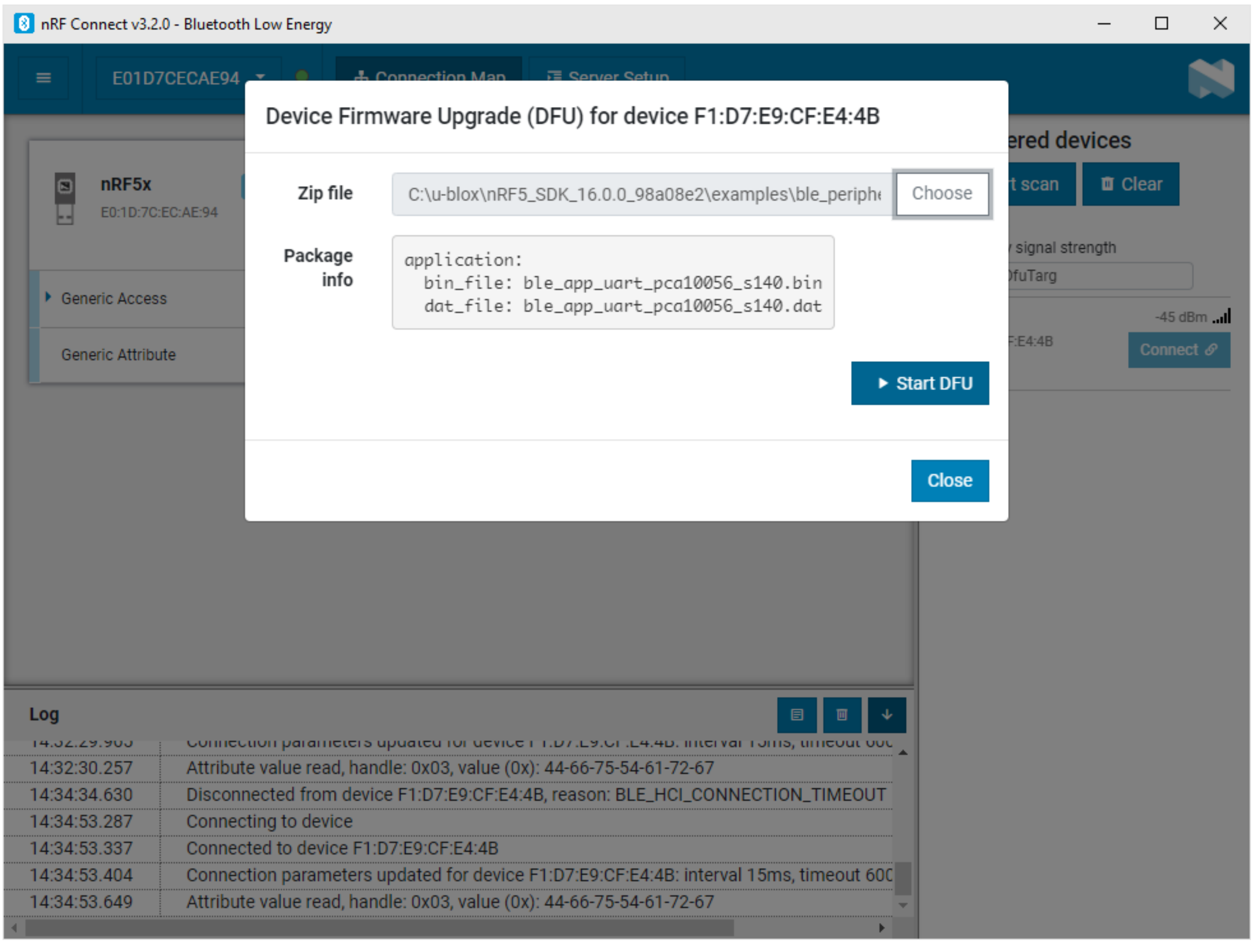
Go ahead and connect, then we'll see the "Secure DFU" service and an update icon:



```
14:31:18.911  Connection parameters updated for device F1:D7:E9:CF:E4:4B: interval 15ms, timeout 60
14:31:19.217  Attribute value read, handle: 0x03, value (0x): 44-66-75-54-61-72-67
14:31:33.993  Disconnected from device F1:D7:E9:CF:E4:4B, reason: BLE_HCI_LOCAL_HOST_TERMINAT
14:32:29.859  Connecting to device
14:32:29.903  Connected to device F1:D7:E9:CF:E4:4B
14:32:29.965  Connection parameters updated for device F1:D7:E9:CF:E4:4B: interval 15ms, timeout 60
14:32:30.257  Attribute value read, handle: 0x03, value (0x): 44-66-75-54-61-72-67
```

Figure 9: nRF Connect showing secure DFU icon

Go ahead and start the DFU process. Navigate to the zip file we just created and click on "Start DFU"



Device Firmware Upgrade (DFU) for device F1:D7:E9:CF:E4:4B

Zip file: C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_periph

Package info:

```
application:
  bin_file: ble_app_uart_pca10056_s140.bin
  dat_file: ble_app_uart_pca10056_s140.dat
```

```
14:32:30.257  Attribute value read, handle: 0x03, value (0x): 44-66-75-54-61-72-67
14:34:34.630  Disconnected from device F1:D7:E9:CF:E4:4B, reason: BLE_HCI_CONNECTION_TIMEOUT
14:34:53.287  Connecting to device
14:34:53.337  Connected to device F1:D7:E9:CF:E4:4B
14:34:53.404  Connection parameters updated for device F1:D7:E9:CF:E4:4B: interval 15ms, timeout 600
14:34:53.649  Attribute value read, handle: 0x03, value (0x): 44-66-75-54-61-72-67
```

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 we started with - though now it also has the bootloader on-board. If we power cycle the Eval board while holding Button 4, then it will re-enter bootloader mode indicated by both LED1 and LED2 being lit.

Up to this point, we've accomplished the following:

- Confirmed that our application code works as expected
- Generated private and public keys for our 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 special consideration. We only needed to generate the DFU zip file from the existing application hex and generated public key files.

5 Buttonless DFU

What if we don't have a button available to enter bootloader mode? Or better yet, how about enhancing our customers' user experience by starting the process automatically? That's where the "buttonless DFU" comes in. This will require some modification to the application to add a new service to enable 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 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 from Nordic Semiconductor:

https://devzone.nordicsemi.com/nordic/short-range-guides/b/bluetooth-low-energy/posts/ble-services-a-beginners-tutorial

In short, this service is performing what we originally enabled with pressing button 4 while resetting the Eval board.

Let's port the ble_app_buttonless_dfu example into to the ble_app_uart example.

To save the work we've done so far, let's create a new example folder:

```
copy C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working_dfu
```

Change directory into the new folder:

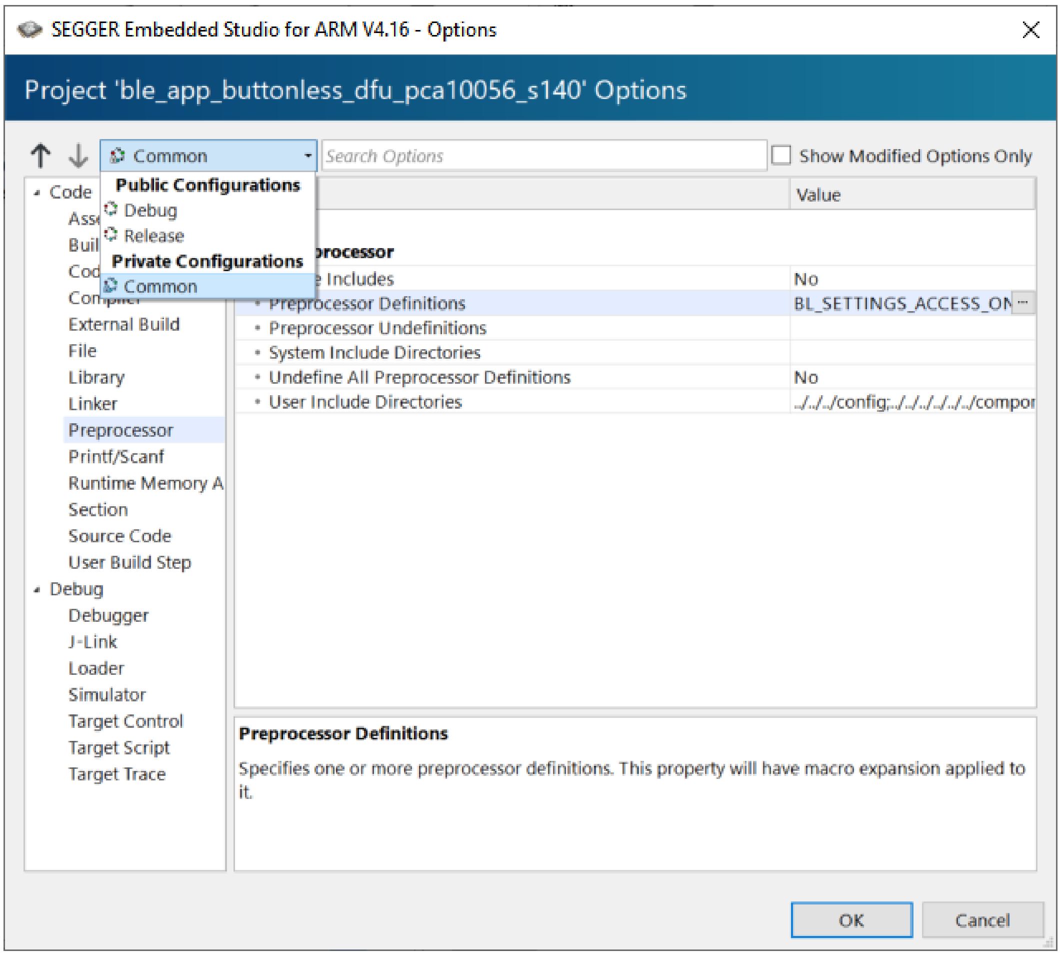
```
cd C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working_dfu
```

Open the project file from here:

```
C:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working_dfu
\pca10056\s140\ses\ble_app_uart_pca10056_s140.emProject
```

5.1 Preprocessor definitions

First we need to configure the project options 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:


SEGGER Embedded Studio for ARM V4.16 - Options

Project 'ble_app_buttonless_dfu_pca10056_s140' Options

	Value
Includes	No
Preprocessor Definitions	BL_SETTINGS_ACCESS_ON
Preprocessor Undefinitions	
System Include Directories	
Undefine All Preprocessor Definitions	No
User Include Directories	../../../config;../../../../../../compor

Preprocessor Definitions

Specifies one or more preprocessor definitions. This property will have macro expansion applied to it.

Figure 11: SES project pre-processor definitions

Add the following to the Preprocessor definitions:

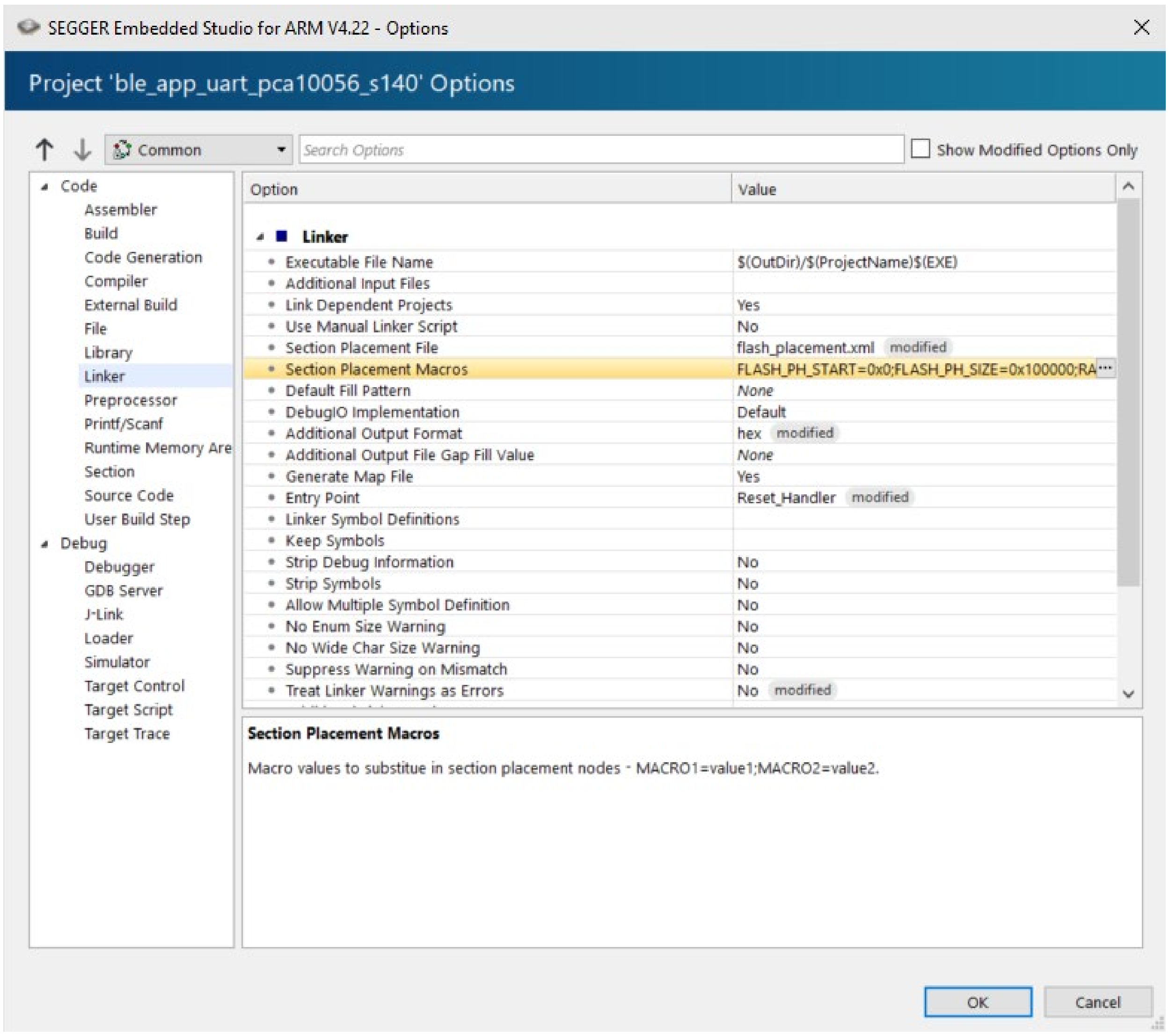
```
BL_SETTINGS_ACCESS_ONLY
NRF_DFU_TRANSPORT_BLE=1
```

And the following 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.)

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


SEGGER Embedded Studio for ARM V4.22 - Options

Project 'ble_app_uart_pca10056_s140' Options

Option	Value
Linker	
Executable File Name	$(OutDir)/$(ProjectName)$(EXE)
Additional Input Files	
Link Dependent Projects	Yes
Use Manual Linker Script	No
Section Placement File	flash_placement.xml modified
Section Placement Macros	FLASH_PH_START=0x0;FLASH_PH_SIZE=0x100000;RA
Default Fill Pattern	*None*
DebugIO Implementation	Default
Additional Output Format	hex modified
Additional Output File Gap Fill Value	*None*
Generate Map File	Yes
Entry Point	Reset_Handler modified
Linker Symbol Definitions	
Keep Symbols	
Strip Debug Information	No
Strip Symbols	No
Allow Multiple Symbol Definition	No
No Enum Size Warning	No
No Wide Char Size Warning	No
Suppress Warning on Mismatch	No
Treat Linker Warnings as Errors	No modified

Section Placement Macros

Macro values to substitue in section placement nodes - MACRO1=value1;MACRO2=value2.

Figure 12: SES project RAM settings

Since we're adding the DFU service alongside the Nordic UART Service (NUS), we need give the new service space in flash. We do this by increasing the start address of RAM and decreasing its size. These are 16 bytes each, so 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
```

5.2 sdk_config.h

Save the options and open sdk_config.h. There are several settings in here to enable DFU. Each line in this list are in separate locations throughout the file. Just search for each of these and change the 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

Now, let's add the necessary library and driver files.

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_16.0.0_98a08e2\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_16.0.0_98a08e2\components\libraries\bootloader\dfu
```

Add this file:

```
nrf_dfu_svci.c
```

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

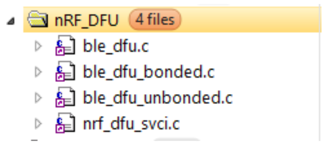


Figure 13: Buttonless UART DFU project - added files

We have the same ble_app_uart functionality as the original example application, though now we also have everything we need to start adding the buttonless DFU function.

5.4 main.c

With the "behind the scenes" items added, we can finally start adding the DFU code to main.c.

Several header files are needed:

```
// 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
```

Let's change the advertising name so we know we're seeing the modified application:

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

Here's where the actual work is done. The first function is the event handler to act on the data being sent to the DFU service.

```
/**@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 second function handles the power management. The bootloader service writes a value to a persistent register, then issues a system reset. 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.

```
/**@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);
```

Here we initialize the DFU service 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

}
```

Go ahead and compile the application, but don't program it to the board just yet. There is a little more prep we need to do.

5.5 Hex file generation

We now have these components:

- Bootloader
- Application with Buttonless DFU code included
- SoftDevice

We're still missing one crucial part - the bootloader settings. These settings values tell the bootloader about the application:

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

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

So we don't have to include long file paths in the nrfutil command, let's copy the hex files we generated to a common directory.

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

Copy the respective hex files to this new location:

```
copy c:\u-blox\nRF5_SDK_16.0.0_98a08e2\components\softdevice\s140\hex\s140_nrf52_7.0.1_softdevice.hex
c:\u-blox\hexfiles

copy c:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\dfu\secure_bootloader\pca10056_s140_ble_wo
rking\ses\Output\Release\Exe\secure_bootloader_ble_s140_pca10056.hex c:\u-blox\hexfiles

copy c:\u-blox\nRF5_SDK_16.0.0_98a08e2\examples\ble_peripheral\ble_app_uart_working\pca100
56\s140\ses\Output\Release\Exe\ble_app_uart_pca10056_s140.hex c:\u-blox\hexfiles

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

Now we can run nrfutil to generate the bootloader settings file:

```
nrfutil settings generate --family NRF52840 --application ble_app_uart_pca10056_s140.hex -
-application-version 0 --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: 0x00000000 (0)
* 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>
```

☞ We used the family of NRF52840 since we're working with the BMD-340. See the `--help` output for other nRF5x versions.

Now we have everything we need to program the BMD-340:

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

Let's merge it all together into a single hex file. The Nordic Semiconductor `mergehex` utility will only allow three input files, so we need two steps:

```
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 try out the new application.

Let's erase the BMD-340 to ensure nothing is left-over from our previous work:

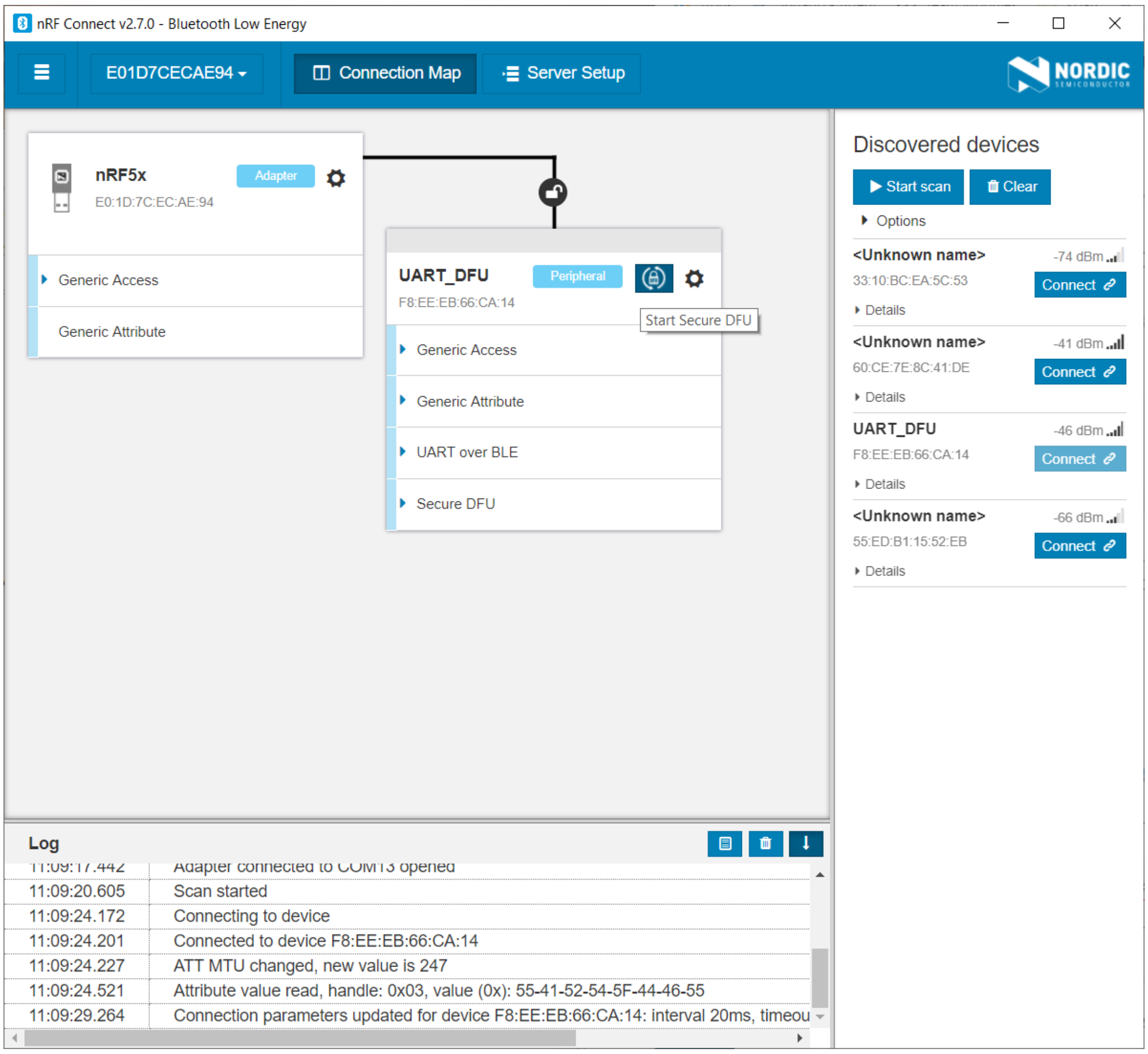
```
c:\u-blox\hexfiles>nrfjprog -f nrf52 --recover
Recovering device. This operation might take 30s.
Erasing user code and UICR flash areas.
```

And, finally, load our combined hex file:

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

Now, power-cycle the BMD-340-EVAL, and the application will start advertising and blink LED1 as before.

Let's connect to it with nRF Connect:



```
11:09:17.442    Adapter connected to COM13 opened
11:09:20.605    Scan started
11:09:24.172    Connecting to device
11:09:24.201    Connected to device F8:EE:EB:66:CA:14
11:09:24.227    ATT MTU changed, new value is 247
11:09:24.521    Attribute value read, handle: 0x03, value (0x): 55-41-52-54-5F-44-46-55
11:09:29.264    Connection parameters updated for device F8:EE:EB:66:CA:14: interval 20ms, timeou
```

Figure 14: nRF Connect with applicatoin showing secure DFU icon

Again, we see the bootloader icon as we did with just the ble_app_buttonless_dfu example, though now we also have the new name, UART_DFU, and the Nordic UART Service. Go ahead and send a few bytes back and forth over the UART service.

We can leave things connected while we prepare a zip file for update.

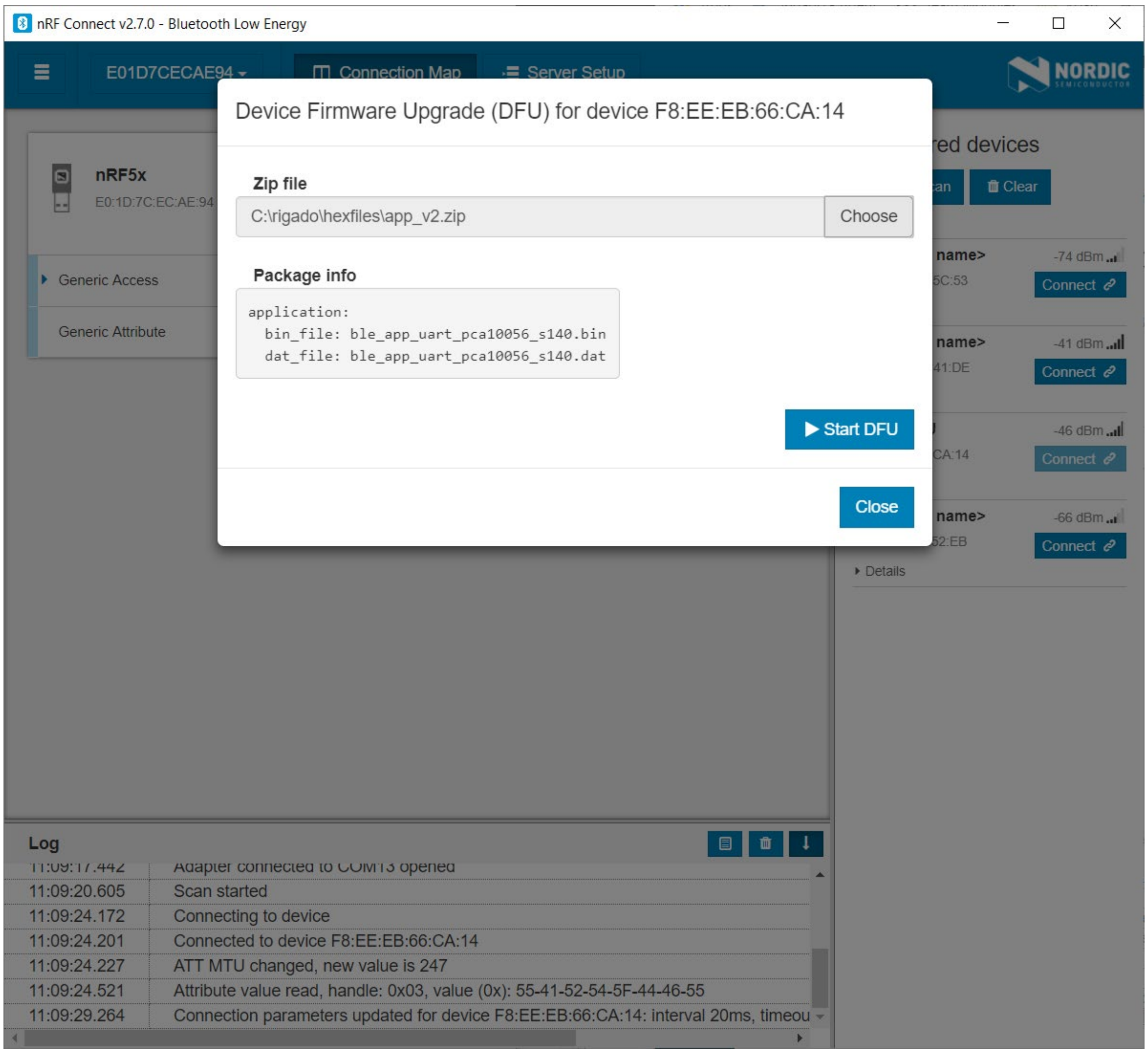
6.2 Prepare and test update

Since we already have the application compiled to a hex file, let's use that as the source application file. In order to inform the bootloader that it's a new, valid update, we'll bump the application version.

In the hexfile directory, use nrfutil again to generate the 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_16.0.0_98a08e2\examples\dfu\dfu_private_key.pem app_v2.zip
```

Go back over to nRF Util and apply the DFU:



```
application:
  bin_file: ble_app_uart_pca10056_s140.bin
  dat_file: ble_app_uart_pca10056_s140.dat
```

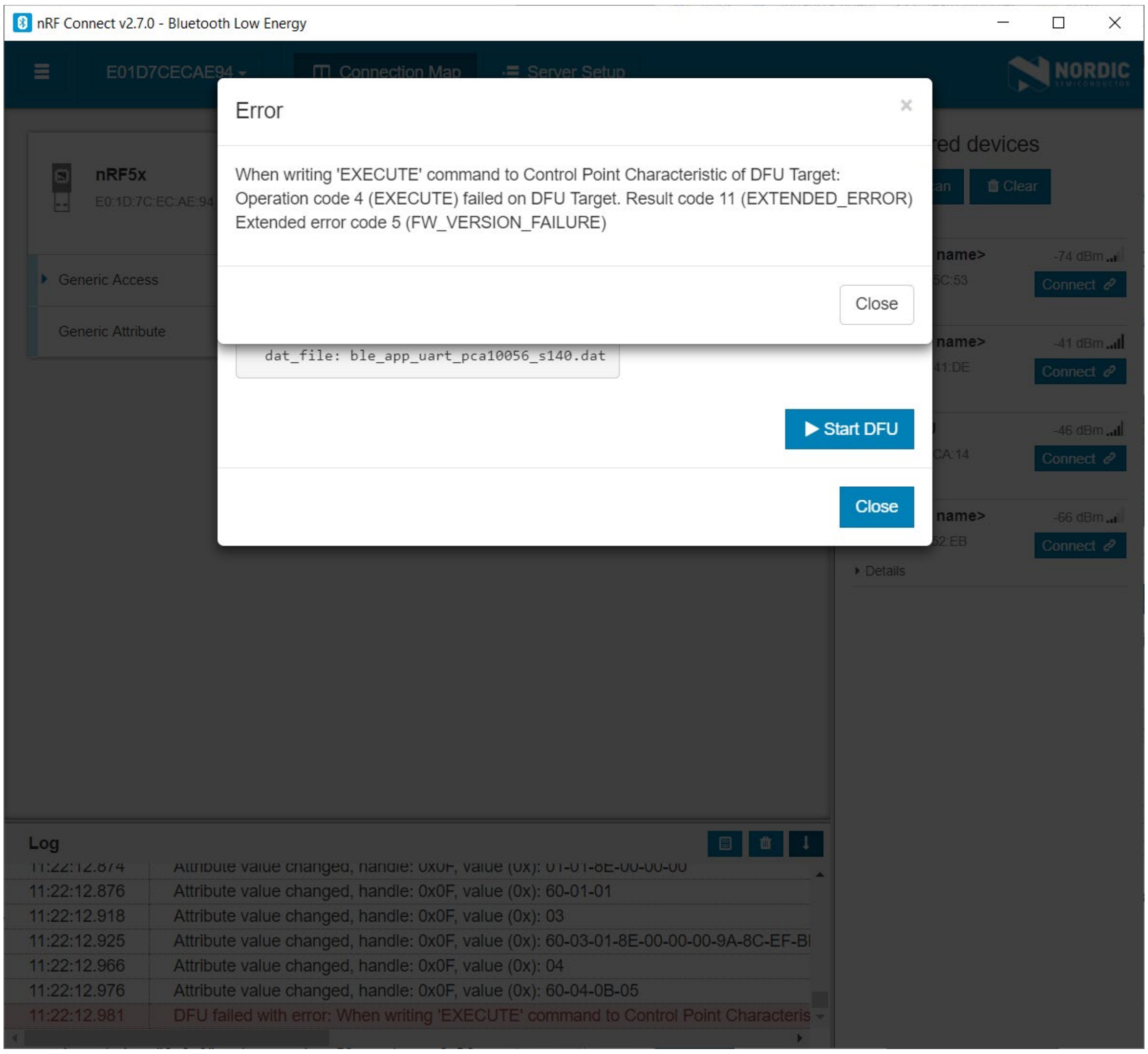
```
11:09:17.442   Adapter connected to COM13 opened
11:09:20.605   Scan started
11:09:24.172   Connecting to device
11:09:24.201   Connected to device F8:EE:EB:66:CA:14
11:09:24.227   ATT MTU changed, new value is 247
11:09:24.521   Attribute value read, handle: 0x03, value (0x): 55-41-52-54-5F-44-46-55
11:09:29.264   Connection parameters updated for device F8:EE:EB:66:CA:14: interval 20ms, timeou
```

Figure 15: SES ready for update from application

To see the version checking in use, let's create another zip file, but use "version 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_16.0.0_98a08e2\examples\dfu\dfu_private_key.pem app_v1.zip
```

If we try to apply it with nRF Connect, we'll see an error since we've already loaded "version 2.0.0".



Error

When writing 'EXECUTE' command to Control Point Characteristic of DFU Target: Operation code 4 (EXECUTE) failed on DFU Target. Result code 11 (EXTENDED_ERROR) Extended error code 5 (FW_VERSION_FAILURE)

```
dat_file: ble_app_uart_pca10056_s140.dat
```

```
11:22:12.874   Attribute value changed, handle: 0x0F, value (0x): 01-01-8E-00-00-00
11:22:12.876   Attribute value changed, handle: 0x0F, value (0x): 60-01-01
11:22:12.918   Attribute value changed, handle: 0x0F, value (0x): 03
11:22:12.925   Attribute value changed, handle: 0x0F, value (0x): 60-03-01-8E-00-00-00-9A-8C-EF-BI
11:22:12.966   Attribute value changed, handle: 0x0F, value (0x): 04
11:22:12.976   Attribute value changed, handle: 0x0F, value (0x): 60-04-0B-05
11:22:12.981   DFU failed with error: When writing 'EXECUTE' command to Control Point Characteris
```

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

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

Thanks to this Nordic Semiconductor DevZone article for inspiration.

There are many more aspects of Nordic's DFU service, such as activating additional transports (UART, USB, Bluetooth Mesh, Zigbee, and Thread), additional security with bond forwarding, etc. For further information, please refer to Nordic Semiconductor's InfoCenter.

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’s 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
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, doc. no. UBX-18009821
[2] NINA-B1 system integration manual, doc. no. UBX-15026175
[3] NINA-B3 system integration manual, doc. no. UBX-17056748
[4] u-blox package information guide, doc. no. UBX-14001652

☞ For regular updates to u-blox documentation and to receive product change notifications, register on our homepage (www.u-blox.com).

Revision history

Revision	Date	Name	Comments
R01	16-Dec-2019	brec	Initial release

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