

Creating a Secure Bootloader Image

451-00004 (BL654-USB for Nordic/Zephyr)

Application Note

v1.0

1 INTRODUCTION

1.1 Scope

The purpose of this document is to guide you through the process of creating a secure bootloader image using Nordic's DFU sample application for the Laird Connectivity 451-00004 – BL654 based USB dongle. This allows for secure signed firmware updates, and to also generate and load a test application.

Note: This is for customers that have a Zephyr/Nordic SDK/other application and want to program it to the USB dongle and prevent unauthorized firmware upgrades. If preventing unauthorized firmware upgrades is not an issue, then the default bootloader can be used as-is.

Note: Laird Connectivity also makes a variant of this product which supports *smartBASIC* development (Laird part # 451-00003). The variants have *different hardware* and are dedicated to their specific development environments.

2 REQUIREMENTS

Before starting, the following are required (see the relevant sections for details on acquiring the components):

- Windows/Linux computer
- Laird Connectivity 451-00004 USB dongle
- Nordic nrfutil command-line utility
- Nordic SDK
- Open bootloader files
- IDE/Compiler

2.1 Windows/Linux Computer

A computer with access to the internet running a Windows operating system (windows 7 or newer) or Linux (4.x kernel or newer) is required. This guide assumes that a Windows computer is being used; commands may be different if running on Linux.

2.2 Laird Connectivity 451-00004 USB Dongle

The Laird Connectivity 451-00004 USB dongle is available world-wide from various distributors. It can be differentiated from the 451-00003 USB dongle (*smartBASIC* dongle) due to a hole in the top part of the case which exposes the reset button (this hole is not found on the 451-00003 dongle). ([Figure 1](#))



Figure 1: Dongle reset button

2.3 Nordic nrfutil Command-Line Utility

Nordic's command line utility *nrfutil* (available for Windows/Linux/Mac) can be used to interact with the module and to create firmware upgrade files from a terminal or command prompt. For windows, a build is available from the [nrfutil Github Releases](#) page; for Linux, follow the instructions listed on the front page of the [nrfutil Github Repository](#).

2.4 Nordic SDK

The Nordic SDK is required to build a secure bootloader image.

Note that this is for the bootloader itself and has no impact on what RTOS/tool the main application is uses. It can be downloaded from the Nordic website: http://developer.nordicsemi.com/nRF5_SDK/ (for this guide, 15.3 was used). Once downloaded, extract the contents of the zip file to a directory that does not have any spaces in.

2.5 Open Bootloader Files

Some files are required for generating the update package. These files can be downloaded from https://github.com/LairdCP/Nordic_DFU_BL654_USB. The repository does not need to be cloned from git; you can download an archive by clicking **Clone or download > Download ZIP** ().

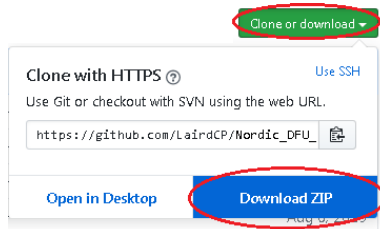


Figure 2: Download an archive

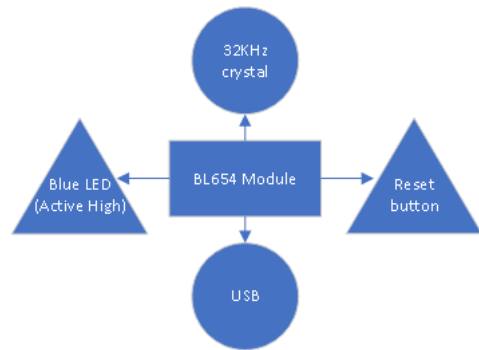
This repository also contains a sample pre-modified secure bootloader project from the 15.3 SDK.

2.6 IDE/Compiler

A C compiler which supports the nRF52840 (Cortex-M4) is required and an optional IDE for editing the project files. The Nordic SDK supports Keil µVision 5, GNU GCC, IAR, and Segger Embedded Studio (SES). This guide was tested with GCC. The GCC build can be downloaded from the ARM site: <https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads> (2019 q3 was used for this guide). Instructions are also provided for Segger Embedded Studio (SES). Installing Segger Embedded Studio (SES) or configuring it is outside the scope of this guide.

3 HARDWARE SETTINGS

By default, the Laird Connectivity 451-00004 module sets the internal high-voltage regulator to run at 3.3v, enables readback protection, and disables CPU patch functionality to protect the integrity of the code running on the module and prevent it being read back via SWD. Because these settings are stored in the UICR, they cannot be re-configured or disabled. There is a 32KHz crystal (± 20 ppm error) connected to the low speed oscillator port of the module and active-high blue LED connected to SIO13 (P0.13).



4 SAMPLE PROJECT

There is a sample secure bootloader project available on github which can be moved into the Nordic SDK version 15.3 under examples\dfu available from https://github.com/LairdCP/Nordic_DFU_BL654_USB which has been pre-modified to use the soft-blinking LED. Note that this project includes modifications for GCC and Segger Embedded Studio (SES) only, for Keil or IAR builds, these changes need to be manually applied.

If you wish to see the changes and manually apply/configure them, follow below. If you use the sample project from github, skip to section 5.8 Adding Public Key to Bootloader.

5 PROJECT SETUP

The secure bootloader project can be located in the Nordic SDK under the following directory:


examples\dfu\secure_bootloader\pca10056_usb

5.1 Updating Flash Location

The bootloader start address must match the start address of the existing bootloader. For SDK 15.3, this is already set correctly but may be set differently if using a different version of the SDK. Check that the start address is set to 0xF4000 and has a length of 0xA000. For the GCC build, this is in **secure_bootloader_gcc_nrf52.ld**.

5.2 Removing LED Code

The secure bootloader code uses multiple LEDs by default which are not present on the 451-00004 dongle hardware so should be removed. Open main.c and remove the bsp_board_* functions:



```

/**
 * @brief Function notifies certain events in DFU process.
 */
static void dfu_observer(nrf_dfu_evt_type_t evt_type)
{
    switch (evt_type)
    {
        case NRF_DFU_EVT_DFU_FAILED:
        case NRF_DFU_EVT_DFU_ABORTED:
        case NRF_DFU_EVT_DFU_INITIALIZED:
            bsp_board_init(BSP_INIT_LEDS);
            bsp_board_led_on(BSP_BOARD_LED_0);
            bsp_board_led_on(BSP_BOARD_LED_1);
            bsp_board_led_off(BSP_BOARD_LED_2);
            break;
        case NRF_DFU_EVT_TRANSPORT_ACTIVATED:
            bsp_board_led_off(BSP_BOARD_LED_1);
            bsp_board_led_on(BSP_BOARD_LED_2);
            break;
        case NRF_DFU_EVT_DFU_STARTED:
            break;
        default:
            break;
    }
}

```

```

/**
 * @brief Function notifies certain events in DFU process.
 */
static void dfu_observer(nrf_dfu_evt_type_t evt_type)
{
    switch (evt_type)
    {
        case NRF_DFU_EVT_DFU_FAILED:
        case NRF_DFU_EVT_DFU_ABORTED:
        case NRF_DFU_EVT_DFU_INITIALIZED:
            break;
        case NRF_DFU_EVT_TRANSPORT_ACTIVATED:
            break;
        case NRF_DFU_EVT_DFU_STARTED:
            break;
        default:
            break;
    }
}

/**@brief Function for application main entry. */
int main(void)
{

```

With the code as-is, there is no longer be any indication via LED that the module is in bootloader mode. This can optionally be changed so that the LED is used for status indication if desired through one of two methods:

- Simple LED on/off – This allows the LED to be turned on and off depending on its state. This requires minimal modifications.
- Soft-blink LED – The pre-loaded bootloader uses this on the dongle to smoothly fade the LED in and out. This requires numerous changes and increases RAM usage and flash consumption.

5.3 Adding Simple LED On/Off (optional)

Inside main.c in the dfu_observer function, add the following code under the NRF_DFU_EVT_DFU_INITIALIZED line to configure the LED and turn it off:

```

nrf_gpio_cfg(13, NRF_GPIO_PIN_DIR_OUTPUT, NRF_GPIO_PIN_INPUT_CONNECT,
NRF_GPIO_PIN_NOPULL, NRF_GPIO_PIN_S0S1, NRF_GPIO_PIN_NOSENSE);

nrf_gpio_pin_write(13, 0);

```

And under the NRF_DFU_EVT_TRANSPORT_ACTIVATED line, add the following code to turn the LED on:

```
nrf_gpio_pin_write(13, 1);
```

The 1 and 0 of the nrf_gpio_pin_write() functions can be switched to have the LED on at bootloader start and turn off when it is initialized, if desired. The complete function for SDK 15.3 looks like the following:

```

static void dfu_observer(nrf_dfu_evt_type_t evt_type)
{
    switch (evt_type)
    {
        case NRF_DFU_EVT_DFU_FAILED:
        case NRF_DFU_EVT_DFU_ABORTED:
        case NRF_DFU_EVT_DFU_INITIALIZED:
            nrf_gpio_cfg(13, NRF_GPIO_PIN_DIR_OUTPUT, NRF_GPIO_PIN_INPUT_CONNECT,
NRF_GPIO_PIN_NOPULL, NRF_GPIO_PIN_S0S1, NRF_GPIO_PIN_NOSENSE);
            nrf_gpio_pin_write(13, 0);
            break;
        case NRF_DFU_EVT_TRANSPORT_ACTIVATED:
            nrf_gpio_pin_write(13, 1);
            break;
        case NRF_DFU_EVT_DFU_STARTED:
            break;
        default:
            break;
    }
}

```

```

/**
 * @brief Function notifies certain events in DFU process.
 */
static void dfu_observer(nrf_dfu_evt_type_t evt_type)
{
    switch (evt_type)
    {
        case NRF_DFU_EVT_DFU_FAILED:
        case NRF_DFU_EVT_DFU_ABORTED:
        case NRF_DFU_EVT_DFU_INITIALIZED:
            nrf_gpio_cfg(13, NRF_GPIO_PIN_DIR_OUTPUT,
                        NRF_GPIO_PIN_INPUT_CONNECT,
                        NRF_GPIO_PIN_NOPULL,
                        NRF_GPIO_PIN_S0S1,
                        NRF_GPIO_PIN_NOSENSE);
            nrf_gpio_pin_write(13, 0);
            break;
        case NRF_DFU_EVT_TRANSPORT_ACTIVATED:
            nrf_gpio_pin_write(13, 1);
            break;
        case NRF_DFU_EVT_DFU_STARTED:
            break;
        default:
            break;
    }
}

/**
 * @brief Function notifies certain events in DFU process.
 */
static void dfu_observer(nrf_dfu_evt_type_t evt_type)
{
    switch (evt_type)
    {
        case NRF_DFU_EVT_DFU_FAILED:
        case NRF_DFU_EVT_DFU_ABORTED:
        case NRF_DFU_EVT_DFU_INITIALIZED:
            nrf_gpio_cfg(13, NRF_GPIO_PIN_DIR_OUTPUT,
                        NRF_GPIO_PIN_INPUT_CONNECT,
                        NRF_GPIO_PIN_NOPULL,
                        NRF_GPIO_PIN_S0S1,
                        NRF_GPIO_PIN_NOSENSE);
            nrf_gpio_pin_write(13, 1);
            break;
        case NRF_DFU_EVT_TRANSPORT_ACTIVATED:
            nrf_gpio_pin_write(13, 0);
            break;
        case NRF_DFU_EVT_DFU_STARTED:
            break;
        default:
            break;
    }
}

```

This concludes enabling the simple on/off LED functionality.

5.4 Adding Soft-Blink LED (optional)

To enable the soft-blink functionality, open the main.c file and add this to the include section:

```

#include "led_softblink.h"
#include "app_timer.h"
#include "nrf_clock.h"

```

Directly below this, add the following defines and application timer definition:

```

#define LAIRD_LED_SIO    13
#define LAIRD_LED        0b1000000000000000    //Bit-mask format

/* Timer used to blink LED on DFU progress. */
APP_TIMER_DEF(m_dfu_progress_led_timer);

```

Directly above the dfu_observer() function, add this timer timeout handler:

```

static void dfu_progress_led_timeout_handler(void * p_context)
{
    app_timer_id_t timer = (app_timer_id_t)p_context;

    uint32_t err_code = app_timer_start(timer,
                                        APP_TIMER_TICKS(DFU_LED_CONFIG_PROGRESS_BLINK_MS),
                                        p_context);

    APP_ERROR_CHECK(err_code);

    nrf_gpio_pin_toggle(LAIRD_LED_SIO);
}

```

Add the following code to the top of the dfu_observer() function:

```

static bool timer_created = false;
uint32_t err_code;

if (!timer_created)
{
    err_code = app_timer_create(&m_dfu_progress_led_timer,
                              APP_TIMER_MODE_SINGLE_SHOT,
                              dfu_progress_led_timeout_handler);

    APP_ERROR_CHECK(err_code);
    timer_created = true;
}

```

Add this code under the NRF_DFU_EVT_DFU_FAILED and NRF_DFU_EVT_DFU_ABORTED case statements:

```
err_code = led_softblink_stop();
APP_ERROR_CHECK(err_code);

err_code = app_timer_stop(m_dfu_progress_led_timer);
APP_ERROR_CHECK(err_code);

err_code = led_softblink_start(LAIRD_LED);
APP_ERROR_CHECK(err_code);
break;
```

Add the following code the NRF_DFU_EVT_DFU_INITIALIZED case statement, before the break; line

```
nrf_gpio_cfg(LAIRD_LED_SIO, NRF_GPIO_PIN_DIR_INPUT, NRF_GPIO_PIN_INPUT_CONNECT,
NRF_GPIO_PIN_NOPULL, NRF_GPIO_PIN_S0S1, NRF_GPIO_PIN_NOSENSE);

if (!nrf_clock_lf_is_running())
{
    nrf_clock_task_trigger(NRF_CLOCK_TASK_LFCLKSTART);
}
err_code = app_timer_init();
APP_ERROR_CHECK(err_code);

led_sb_init_params_t led_sb_init_param = LED_SB_INIT_DEFAULT_PARAMS(LAIRD_LED);

uint32_t ticks = APP_TIMER_TICKS(DFU_LED_CONFIG_TRANSPORT_INACTIVE_BREATH_MS);
led_sb_init_param.p_leds_port = BSP_LED_1_PORT;
led_sb_init_param.on_time_ticks = ticks;
led_sb_init_param.off_time_ticks = ticks;
led_sb_init_param.duty_cycle_max = 255;
led_sb_init_param.active_high = true;

err_code = led_softblink_init(&led_sb_init_param);
APP_ERROR_CHECK(err_code);

err_code = led_softblink_start(LAIRD_LED);
APP_ERROR_CHECK(err_code);
```

Add the following under the NRF_DFU_EVT_TRANSPORT_ACTIVATED case statement:

```
{
    uint32_t ticks = APP_TIMER_TICKS(DFU_LED_CONFIG_TRANSPORT_ACTIVE_BREATH_MS);
    led_softblink_off_time_set(ticks);
    led_softblink_on_time_set(ticks);
}
```

Add the following case statement after the end of the NRF_DFU_EVT_TRANSPORT_ACTIVATED case statement:

```
case NRF_DFU_EVT_TRANSPORT_DEACTIVATED:
{
    uint32_t ticks = APP_TIMER_TICKS(DFU_LED_CONFIG_PROGRESS_BLINK_MS);
    err_code = led_softblink_stop();
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_start(m_dfu_progress_led_timer, ticks,
m_dfu_progress_led_timer);
    APP_ERROR_CHECK(err_code);
    break;
}
```

Add in this code to sdk_config.h to enable the required features:

```
// <q> LED_SOFTBLINK_ENABLED - led_softblink - led_softblink module

#ifndef LED_SOFTBLINK_ENABLED
#define LED_SOFTBLINK_ENABLED 1
#endif

// <q> LOW_POWER_PWM_ENABLED - low_power_pwm - low_power_pwm module

#ifndef LOW_POWER_PWM_ENABLED
#define LOW_POWER_PWM_ENABLED 1
#endif

// <e> APP_TIMER_ENABLED - app_timer - Application timer functionality
//=====
#ifndef APP_TIMER_ENABLED
#define APP_TIMER_ENABLED 1
#endif
// <o> APP_TIMER_CONFIG_RTC_FREQUENCY - Configure RTC prescaler.

// <0=> 32768 Hz
// <1=> 16384 Hz
// <3=> 8192 Hz
// <7=> 4096 Hz
// <15=> 2048 Hz
// <31=> 1024 Hz

#ifndef APP_TIMER_CONFIG_RTC_FREQUENCY
#define APP_TIMER_CONFIG_RTC_FREQUENCY 1
#endif

// <o> APP_TIMER_CONFIG_IRQ_PRIORITY - Interrupt priority

// <i> Priorities 0,2 (nRF51) and 0,1,4,5 (nRF52) are reserved for SoftDevice
// <0=> 0 (highest)
// <1=> 1
// <2=> 2
// <3=> 3
// <4=> 4
// <5=> 5
// <6=> 6
// <7=> 7

#ifndef APP_TIMER_CONFIG_IRQ_PRIORITY
#define APP_TIMER_CONFIG_IRQ_PRIORITY 6
#endif

// <o> APP_TIMER_CONFIG_OP_QUEUE_SIZE - Capacity of timer requests queue.
// <i> Size of the queue depends on how many timers are used
// <i> in the system, how often timers are started and overall
// <i> system latency. If queue size is too small app_timer calls
// <i> will fail.

#ifndef APP_TIMER_CONFIG_OP_QUEUE_SIZE
#define APP_TIMER_CONFIG_OP_QUEUE_SIZE 10
#endif

// <q> APP_TIMER_CONFIG_USE_SCHEDULER - Enable scheduling app_timer events to
app_scheduler
```

```
#ifndef APP_TIMER_CONFIG_USE_SCHEDULER
#define APP_TIMER_CONFIG_USE_SCHEDULER 0
#endif

// <q> APP_TIMER_KEEPS_RTC_ACTIVE - Enable RTC always on

// <i> If option is enabled RTC is kept running even if there is no active timers.
// <i> This option can be used when app_timer is used for timestamping.

#ifndef APP_TIMER_KEEPS_RTC_ACTIVE
#define APP_TIMER_KEEPS_RTC_ACTIVE 0
#endif

// <o> APP_TIMER_SAFE_WINDOW_MS - Maximum possible latency (in milliseconds) of handling
app_timer event.
// <i> Maximum possible timeout that can be set is reduced by safe window.
// <i> Example: RTC frequency 16384 Hz, maximum possible timeout 1024 seconds -
APP_TIMER_SAFE_WINDOW_MS.
// <i> Since RTC is not stopped when processor is halted in debugging session, this value
// <i> must cover it if debugging is needed. It is possible to halt processor for
APP_TIMER_SAFE_WINDOW_MS
// <i> without corrupting app_timer behavior.

#ifndef APP_TIMER_SAFE_WINDOW_MS
#define APP_TIMER_SAFE_WINDOW_MS 300000
#endif

// <q> NRF_SORTLIST_ENABLED - nrf_sortlist - Sorted list

#ifndef NRF_SORTLIST_ENABLED
#define NRF_SORTLIST_ENABLED 1
#endif

//=====
// <h> Bootloader LEDs Configuration
//=====
// <o> DFU_LED_CONFIG_TRANSPORT_ACTIVE_BREATH_MS - Active and Inactive period (in
milliseconds) of LED breathing when DFU transport is active (e.g. BLE connected).
#ifndef DFU_LED_CONFIG_TRANSPORT_ACTIVE_BREATH_MS
#define DFU_LED_CONFIG_TRANSPORT_ACTIVE_BREATH_MS 300
#endif

// <o> DFU_LED_CONFIG_TRANSPORT_INACTIVE_BREATH_MS - Active and Inactive period (in
milliseconds) of LED breathing when DFU transport is inactive (e.g. BLE disconnected).
#ifndef DFU_LED_CONFIG_TRANSPORT_INACTIVE_BREATH_MS
#define DFU_LED_CONFIG_TRANSPORT_INACTIVE_BREATH_MS 600
#endif

// <o> DFU_LED_CONFIG_PROGRESS_BLINK_MS - Active and Inactive period of LED blinking when
DFU progress is ongoing.
#ifndef DFU_LED_CONFIG_PROGRESS_BLINK_MS
#define DFU_LED_CONFIG_PROGRESS_BLINK_MS 100
#endif }
```


5.5 Adding Soft-Blink LED includes/defines (optional)

Soft-blink LED status can be used to indicate the status of the bootloader and will make the LED softly fade in and out when the device is in bootloader mode.

5.5.1 GCC

Add these extra C files:

```
$(SDK_ROOT)/components/libraries/sortlist/nrf_sortlist.c \  
$(SDK_ROOT)/components/libraries/timer/experimental/app_timer2.c \  
$(SDK_ROOT)/components/libraries/timer/experimental/drv_rtc.c \  
$(SDK_ROOT)/components/libraries/led_softblink/led_softblink.c \  
$(SDK_ROOT)/components/libraries/low_power_pwm/low_power_pwm.c \  
$(SDK_ROOT)/components/libraries/atomic_fifo/nrf_atfifo.c \
```

Add these extra include directories:

```
$(SDK_ROOT)/components/libraries/sortlist \  
$(SDK_ROOT)/components/libraries/low_power_pwm \  
$(SDK_ROOT)/components/libraries/led_softblink \  
$(SDK_ROOT)/components/libraries/timer/experimental \  
$(SDK_ROOT)/components/libraries/timer \  
$(SDK_ROOT)/components/libraries/atomic_fifo \
```

Add the following to the C flags:

```
CFLAGS += -DAPP_TIMER_V2  
CFLAGS += -DAPP_TIMER_V2_RTC1_ENABLED
```

Add the following to the assembly flags:

```
ASMFLAGS += -DAPP_TIMER_V2  
ASMFLAGS += -DAPP_TIMER_V2_RTC1_ENABLED
```

The linker file for the secure bootloader in SDK 15.3 is wrong and needs fixing, open `secure_bootloader_gcc_nrf52.ld` and change:

```
uicr_bootloader_start_address (r) : ORIGIN = 0x00000FF8, LENGTH = 0x4
```

To:

```
uicr_bootloader_start_address (r) : ORIGIN = 0x10001014, LENGTH = 0x4
```

Then change:

```
uicr_mbr_params_page (r) : ORIGIN = 0x00000FFC, LENGTH = 0x4
```

To:

```
uicr_mbr_params_page (r) : ORIGIN = 0x10001018, LENGTH = 0x4
```

5.5.2 Segger Embedded Studio (SES)

With the SES project open, right click on the project on the left-hand side project view and select **options**. In the drop-down menu, select **common** and search for **include**. Then append the following to the end of the User Include Directories option:

```
../../../../components/libraries/sortlist
../../../../components/libraries/low_power_pwm
../../../../components/libraries/led_softblink
../../../../components/libraries/timer/experimental
../../../../components/libraries/timer
```

Click **OK** and search for **Preprocessor**. Then append the following to the end of the Preprocessor Definitions option:

```
APP_TIMER_V2
APP_TIMER_V2_RTC1_ENABLED
```

Click **OK** and click **OK** again to apply the settings.

Right click on the *nRF_Drivers* folder in the left-hand side project view and select **Add existing files**. Then add the following file, relative to your nRF SDK v15.3 root directory:

```
components/libraries/timer/experimental/drv_rtc.c
```

Right click on the *nRF_Libraries* folder in the left-hand side project view and select **Add existing files**. Add the following files one-by-one, relative to your nRF SDK v15.3 root directory:

```
components/libraries/sortlist/nrf_sortlist.c
components/libraries/timer/experimental/app_timer2.c
components/libraries/led_softblink/led_softblink.c
components/libraries/low_power_pwm/low_power_pwm.c
components/libraries/atomic_fifo/nrf_atfifo.c
```

The linker file for the secure bootloader in SDK 15.3 is wrong and must be fixed. To do this, right-click on the project in the left-hand side project view and select **Edit Section Placement**, in the xml file that opens change:

```
<MemorySegment name="uicr_bootloader_start_address" start="0x00000FF8" size="0x4">
  <ProgramSection alignment="4" keep="Yes" load="Yes"
name=".uicr_bootloader_start_address"
address_symbol="__start_uicr_bootloader_start_address"
end_symbol="__stop_uicr_bootloader_start_address" start = "0x00000FF8" size="0x4" />
```

To:

```
<MemorySegment name="uicr_bootloader_start_address" start="0x10001014" size="0x4">
  <ProgramSection alignment="4" keep="Yes" load="Yes"
name=".uicr_bootloader_start_address"
address_symbol="__start_uicr_bootloader_start_address"
end_symbol="__stop_uicr_bootloader_start_address" start = "0x10001014" size="0x4" />
```

Then change:

```
<MemorySegment name="uicr_mbr_params_page" start="0x00000FFC" size="0x4">
  <ProgramSection alignment="4" keep="Yes" load="Yes" name=".uicr_mbr_params_page"
address_symbol="__start_uicr_mbr_params_page" end_symbol="__stop_uicr_mbr_params_page"
start = "0x00000FFC" size="0x4" />
```

To:

```
<MemorySegment name="uicr_mbr_params_page" start="0x10001018" size="0x4">
  <ProgramSection alignment="4" keep="Yes" load="Yes" name=".uicr_mbr_params_page"
address_symbol="__start_uicr_mbr_params_page" end_symbol="__stop_uicr_mbr_params_page"
start = "0x10001018" size="0x4" />
```

5.6 Updating SDK Configuration

By default, the secure bootloader can be entered by having a GPIO set at start-up or by setting the GPREGRET register and restarting the module. However, on the Laird dongle bootloader, the bootloader is entered by pressing the reset button, this ensures that even if an application with a fault is loaded then the module can be recovered. To switch to using the reset button to enter the bootloader, open the `sdk_config.h` file and search for this line:

```
#define NRF_BL_DFU_ENTER_METHOD_BUTTON 1
```

Change the 1 to a 0 to prevent using a button to enter DFU mode like so:

```
#define NRF_BL_DFU_ENTER_METHOD_BUTTON 0
```

Then search for this line:

```
#define NRF_BL_DFU_ENTER_METHOD_PINRESET 0
```

Change the 0 to a 1 like so to enable using the reset button to enter DFU mode:

```
#define NRF_BL_DFU_ENTER_METHOD_PINRESET 1
```

5.7 Generating a Public/Private Keypair

For the signed operation of the bootloader to work, a private/public keypair must be generated. The purpose of the private key is to sign some data (in this instance, a fixed size checksum/hash of the application) which can then be sent with firmware upgrade packages. The public key gets embedded into the bootloader and is used to verify signatures but cannot generate them. A private key can be used to calculate a public key, but a public key cannot be used to calculate a private key, this provides a secure firmware upgrade mechanism.

It is imperative that the private key be stored safely. If the key is lost, then all units that have been programmed cannot be upgraded again via the bootloader; and if the key is exposed or leaked, it can then be used to sign fake firmware files which could (depending on application) lead to information loss or far reaching security problems.

The `nrfutil` program is used to generate a keypair. It can be generated using the following command; the output filename can be set by adjusting the `user_key.pem` command line argument:

```
nrfutil.exe keys generate user_key.pem
```

This file contains the private and public keys which can be viewed respectively by issuing the commands:

```
nrfutil.exe keys display --key sk --format hex user_key.pem
nrfutil.exe keys display --key pk --format hex user_key.pem
```

5.8 Adding Public Key to Bootloader

Use `nrfutil` to generate a `.c` file which contains the public key by running:

```
nrfutil.exe keys display --key pk --format code user_key.pem > dfu_public_key.c
```

If this is not done in the examples/dfu directory, then the file must be moved there (and overwrite the existing file). Once moved, this becomes the public key which is included in the secure bootloader.

5.9 Building Project

Using the IDE/supported toolchain of your choice, compile/rebuild the project to generate an output hex file. If using GCC, this is performed by going to the **examples\dfu\secure_bootloader\pca10056_usb\armgcc** folder and issuing the command:

```
make -j3
```

The `-j3` argument is used to build using three parallel threads which speeds up compilation on multi-core/CPU systems. It can be omitted if building on a single core system or increased, if desired.

If using Segger Embedded Studio (SES), ensure you have the 'Release' build selected in the dropdown on the left-hand side project view and click the 'Build' menu and select the 'Build solution' option.

6 CREATING SIGNED UPGRADE PACKAGE

6.1 GCC

The compiled application hex file is located inside the `_build` directory. This can be turned into a firmware upgrade package by using `nrfutil`, ensure that the `laird_dongle_public.pem` file from the Laird Connectivity open bootloader repository is moved into this directory and issue the command:

```
nrfutil pkg generate bl_upd.zip --bootloader nrf52840_xxaa.hex --bootloader-version 1 --  
hw-version 52 --sd-req 0x00 --key-file laird_dongle_public.pem
```

6.2 Segger Embedded Studio (SES)

The compiled application hex file is located inside the `Output\Release\Exe` directory. This can be turned into a firmware upgrade package by using `nrfutil`, ensure that the `laird_dongle_public.pem` file from the Laird Connectivity open bootloader repository is moved into this directory and issue the command:

```
nrfutil pkg generate bl_upd.zip --bootloader secure_bootloader_usb_mbr_pca10056.hex --  
bootloader-version 1 --hw-version 52 --sd-req 0x00 --key-file laird_dongle_public.pem
```

6.3 Version Note

The existing bootloader has a version of 0, therefore 1 is the version of the bootloader which needs to be used to upgrade. The hardware version of 52 indicates it is for the nRF52 chipset; the SD requirement of 0 indicates that the softdevice is not required/used. Once ran, the output package `bl_upd.zip` is created which can be used to upgrade the bootloader on the modules using `nrfutil`.

7 UPGRADE DONGLE TO SECURE BOOTLOADER

To enter bootloader mode, plug in a Laird Connectivity 451-00004 USB dongle and press the reset button. The LED fades in and out to indicate that it is in bootloader mode. You may need to install drivers for the USB serial port which are located in the NRF SDK folder directory **examples\usb_drivers**.

With the dongle in bootloader mode, upgrade the unit to the new secure bootloader inside the signed firmware zip file by running the command (substituting COM87 for the serial port of the plugged-in dongle, check device manager to find this out if you do not know it):

```
nrfutil.exe dfu usb-serial -pkg bl_upd.zip -p COM87
```

The process should complete successfully, and the dongle will reboot to run the new bootloader. Before rolling this out to all dongles, the programmed image should first be tested to ensure it works by programming a user application to it to ensure that the correct signing key was used. This is explained in the following sections.

8 BUILDING A SAMPLE USER APPLICATION

To test that the public key used for verifying signatures has been entered successfully, we recommend that you build a sample application and test it to ensure it can be programmed to a module using the bootloader. For this guide, the BLE HRS (heart rate service) sample application in the Nordic SDK is used but any application in the Nordic SDK or Zephyr SDK can be used.

The HRS peripheral application is located in the `examples\ble_peripheral\ble_app_hrs` folder and the `pca10059` build is used. Navigate to the `examples\ble_peripheral\ble_app_hrs\pca10059\140` directory and build the application using your chosen compiler, which for GCC is achieved by running `make` in the `armgcc` folder or for Segger Embedded Studio requires using the project file in the `SES` folder. Once building has finished, a hex file is located in the `_build` sub-folder (GCC) or `Output\Release\Exe` sub-folder (SES).

Note: If using Zephyr, the application must start at offset 0x1000, not 0x0. This can be set by configuring `FLASH_LOAD_OFFSET` to 0x1000 before building the application.
In the GUI configuration, this option is under **Build and Link features > Linker Options > Kernel load offset**.

9 BUILDING SAMPLE APPLICATION FIRMWARE UPDATE PACKAGE

Similar to how the bootloader upgrade package is built, `nrfutil` is used to generate an application update package which has separate command line arguments for the user application and softdevice (If using a Zephyr, there is no softdevice file).

9.1 GCC

Run this command for generating a package. This assumes SDK 15.3 and softdevice 6.1.1. Check the SDK files for details/file paths if using a different version. This also assumes `user_key.pem` was copied to the `_build` directory:

```
nrfutil.exe pkg generate --application nrf52840_xxaa.hex --hw-version 52 --application-  
version 1 --sd-req 0xb6,0x00 --sd-id 0xb6 --softdevice  
.....\components\softdevice\s140\hex\s140_nrf52_6.1.1_softdevice.hex --key-  
file user_key.pem app_upd.zip
```

9.2 Segger Embedded Studio

Run this command for generating a package. This assumes SDK 15.3 and softdevice 6.1.1. Check the SDK files for details/file paths if using a different version. This also assumes `user_key.pem` was copied to the `_build` directory:

```
nrfutil.exe pkg generate --application ble_app_hrs_pca10059_s140.hex --hw-version 52 --  
application-version 1 --sd-req 0xb6,0x00 --sd-id 0xb6 --softdevice  
.....\components\softdevice\s140\hex\s140_nrf52_6.1.1_softdevice.hex -  
-key-file user_key.pem app_upd.zip
```

10 PROGRAMMING SAMPLE USER APPLICATION UPDATE PACKAGE

Plug the Laird Connectivity 451-00004 dongle into the computer and ensure it is in bootloader mode by pressing the reset button. Run the following command to program the application and softdevice to the module (substituting COM87 for the serial port of your dongle):

```
nrfutil dfu usb-serial -pkg app_upd.zip -p COM87
```

If an error displays, it could be caused by an incorrect bootloader public key. Check that it was correctly imported then rebuild and try again.

11 TESTING THE SAMPLE USER APPLICATION

Once the module has been programmed, it loads the application and begins advertising. Using a mobile phone or other BLE equipped device, start a scan for advertising modules. The following was tested on iOS using the [nRF connect](#) app.

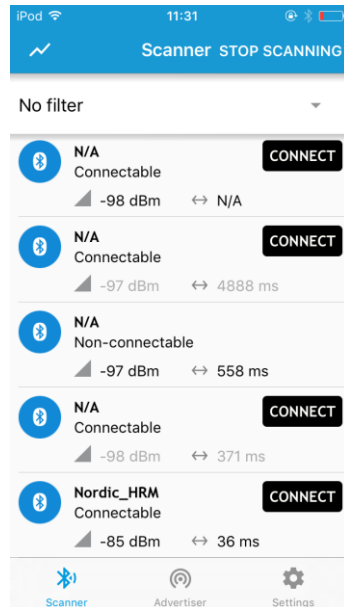


Figure 3: Start a scan

There should be a device advertising with the name *Nordic_HRM*. Connect to the device and ensure it is able to view the GATT table:

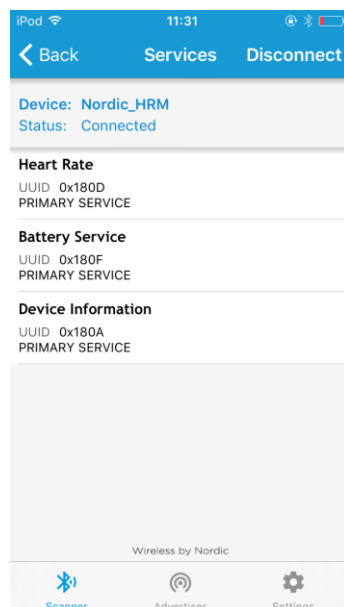


Figure 4: Nordic_HRM device displays

Press the reset button on the dongle and ensure that the module enters bootloader mode. The mobile device should indicate that it has been disconnected:

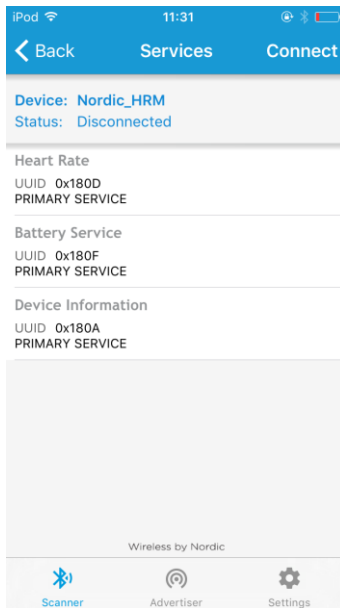


Figure 5: Disconnected mobile device

This concludes testing of the secure bootloader image.

12 ENTERING BOOTLOADER MODE

As well as using the reset button to enter bootloader mode, it can also be entered by setting the GPREGRET register to 0xB1 (equivalent to BOOTLOADER_DFU_START from nrf_bootloader_info.h) and restarting the module using NVIC_SystemReset(), for example.

13 USER APPLICATION CONSTRAINTS

The bootloader and Nordic MBR occupy flash space on the module with the following memory structure:

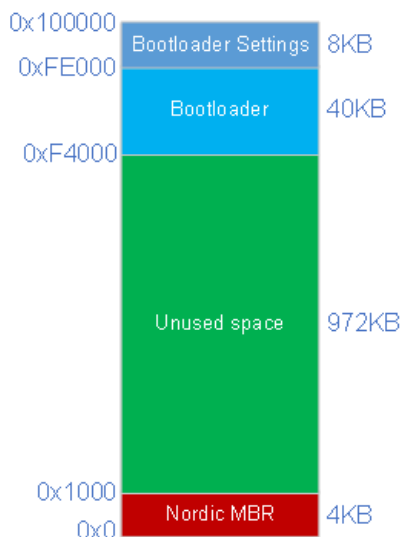


Figure 6: Memory structure

It is important that the linker file of your application is aware of the bootloader and does not try to place code there. It is also important that your application does not try to read/write/erase code at addresses used by the bootloader or bootloader settings.

The bootloader supports single and dual bank firmware updates. This can be tweaked to enable/disable them (by default, dual bank updating is forced which means that the largest application that can be upgraded via the bootloader is $\frac{972\text{KB}}{2} = 486\text{KB}$). Information on what single/dual bank firmware updates are is available on the [Nordic site](#). Control of this is handled in the sdk_config.h file with the following defines:

```
#define NRF_DFU_FORCE_DUAL_BANK_APP_UPDATES 0
#define NRF_DFU_SINGLE_BANK_APP_UPDATES 0
```

The bootloader uses all of the free space for storing upgrade data which begins at the end of the application space. Optionally, if the application requires storage space then it can do this by storing data at 0xF3000 and prior by setting DFU_APP_DATA_RESERVED in nrf_dfu_types.h to a multiple of the page size. So, if this is set to 0x1000, it reserves 0xF2000-0xF3000 for application use; if set to 0x3000, it reserves 0xF0000-0xF3000 for application use. By default, this is set to 0. Further details are available on the [Nordic site](#).

14 REVISION HISTORY

| Version | Date | Notes | Contributor(s) | Approver |
|---------|-------------|-----------------|----------------|---------------|
| 1.0 | 01 Oct 2019 | Initial Release | Jamie McCrae | Jonathan Kaye |