

# How to generate the INIT file for the DFU

## Revision History:

12.11.2019 – Introduction of nrf util, links to documentation updated.

17.03.2015 – Aligning with the DFU from SDK 8.0, adding the Distribution packet part

05.02.2015 – Link to documentation updated, small bugs fixed

15.02.2014 – Document created

## INTRO

Use **nrf util** to create ZIP files that can be used for DFU: <https://github.com/NordicSemiconductor/pc-nrfutil>

For legacy DFU use branch **0\_5\_x** with the latest version.

For Secure DFU use the master branch.

Both versions cannot be installed at the same time on one PC (without modification the source code), as they use the same packet names.

## SECURE DFU

For details, see documentation for the nrf util on GitHub and on:

[https://infocenter.nordicsemi.com/topic/sdk\\_nrf5\\_v16.0.0/lib\\_bootloader\\_dfu\\_validation.html?cp=5\\_1\\_3\\_5\\_1\\_1](https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/lib_bootloader_dfu_validation.html?cp=5_1_3_5_1_1)

## LEGACY DFU

Starting from the DFU Bootloader SDK 7.0 the init packet is required to perform DFU operation. Before, the init packet was optional and could have contained just 2 bytes of the firmware CRC.

The extended init packet has the following syntax:

Device Type	Device Revision
Application Version	
SoftDevice array length (n)	SoftDevice - 1
SoftDevice - 2	.....
SoftDevice - n	CRC – 16 - CCITT
<div><div></div><div>2 bytes</div><div></div><div>2 bytes</div><div></div></div>	

Fig 1.

[http://developer.nordicsemi.com/nRF51\\_SDK/nRF51\\_SDK\\_v8.x.x/doc/8.0.0/s110/html/a00093.html](http://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v8.x.x/doc/8.0.0/s110/html/a00093.html)

[https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.sdk51.v10.0.0/bledfu\\_example\\_image.html?cp=5\\_12\\_4\\_3\\_1\\_1\\_2](https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.sdk51.v10.0.0/bledfu_example_image.html?cp=5_12_4_3_1_1_2)

The init file is a binary file, written in Little Endian.

Device type and revision must match the preprogrammed values on the chip. By default they are set to 0xFFFF and 0xFFFF. The application version currently is ignored (it is up to the customers to validate it, if required), but should be set to 0xFFFFFFFF.

After the application version the init file specifies the list of soft devices compatible with the new firmware. Those values must be correct as this is validated by the bootloader. First 2 bytes specify the length of the list. Remember about the reversed order (little endian). The correct values are:

0100 FFFF – any Soft Device (for development purposes only)  
0100 5A00 – compatible with the SD 7.1.0  
0200 4F00 5A00 – compatible with both SD 7.0.0 and SD 7.1.0  
0100 6400 – compatible with SD 8.0 only  
etc.

Soft Device ids are specified at the bottom of the website given above.

## DISTRIBUTION PACKETS (ZIP)

The upgrade should be distributed in ZIP files created using **nrf util** tool:

Secure DFU: <https://github.com/NordicSemiconductor/pc-nrfutil>

Legacy DFU: [https://github.com/NordicSemiconductor/pc-nrfutil/tree/0\\_5\\_3](https://github.com/NordicSemiconductor/pc-nrfutil/tree/0_5_3)

The **dfu** module may be used to perform multiple operations for DFU:

- Creating distribution packets,
- Converting HEX into BIN files,
- Uploading the firmware over DFU OTA
- Etc.

---

## GENERATING PACKETS USING NRF UTILITY – LEGACY DFU

To create an image file (distribution packet) you must have your application compiled to HEX or BIN format.

Use the following command in the command line:

```
nrf dfu genpkg --application [application].hex --application-version [version] --softdevice [softdevice].hex  
--bootloader [bootloader].hex --dev-type [type] --dev-revision [revision]  
--sd-req [list of supported Soft Device ids] [name of the distribution packet].zip
```

Details:

**--application** – the parameter specifies the name of the application file  
**--softdevice** – the parameter specifies the name of the Soft Device file  
**--bootloader** – the parameter specifies the name of the bootloader image.  
At least one of those files must be specified.

**--application-version** – this parameter is required. The default DFU bootloader implementation does not validate the application version, as specified in the documentation (first link). In this case this value should be set to 0xffffffff: [...] --application-version 0xffffffff [...]

**--dev-type** – the device type. This value is validated against the value in the UICR register. By default it is set to 0xffff. This parameter is required.

**--dev-revision** – the device revision. This value is validated against the UICR register. By default it is set to 0xffff. This parameter is required.

**--sd-req** – a list of supported Soft Devices. This parameter is required. Use the colon to separate supported Soft Devices, e.g. [...] --sd-req 0x4f,0x5a [...]

The DAT file created with this command should have the syntax specified by Figure 1.

Find detailed information about the **dfu** component or the **genpkg** command using:

**nrf dfu --help**

**nrf dfu genpkg --help**

The documentation is also available online here:

[http://developer.nordicsemi.com/nRF51\\_SDK/nRF51\\_SDK\\_v8.x.x/doc/8.0.0/s110/html/a00092.html](http://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v8.x.x/doc/8.0.0/s110/html/a00092.html)

With the **nrf** tool you may create images that contain only an application, SoftDevice or bootloader or a combination of these files. When a SoftDevice and a bootloader firmware files have been specified, the distribution packet will contain a merged BIN file for both of them. This tool automatically converts HEX files to BIN and generates the DAT files.

You may also use the **--dfu-ver** parameter to specify your DFU bootloader version. Currently versions **0.5** and **0.6** are supported. The DAT files generated with both those versions are the same.

## EXAMPLE FILES

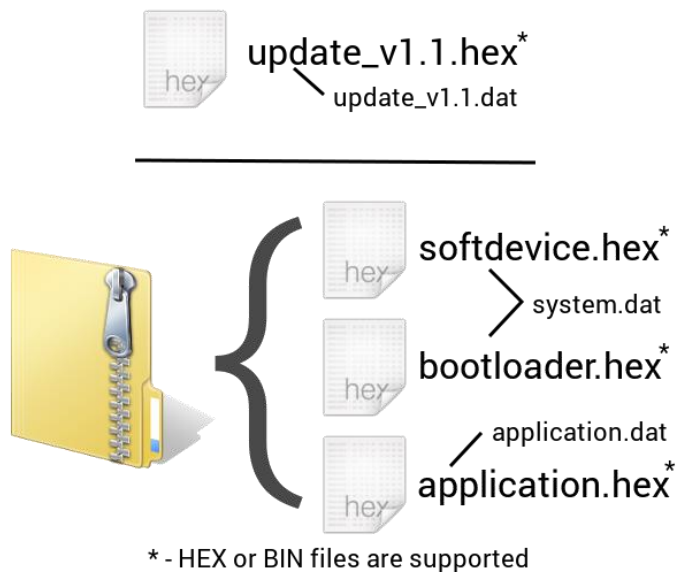
An example init packet (DAT file) for Legacy DFU:

*ffff ffff ffff ffff 0200 4f00 5a00 [CRC]*

## DEPRECATED

Using the **nrf** utility, described above, is a recommended method for creating image files that may contain an Application, SoftDevice and a Bootloader in a single file. For backward compatibility the older methods of distributing firmware updates are also supported. However, they are now deprecated and support for them may end without a notice. The other methods are:

- Specifying a pair of files: a firmware image (HEX or BIN) and, optionally, the DAT file with init data.
- A ZIP file with the fixed naming convention. The **nrf** tool uses the *manifest.json* file to describe the packet content. Before the ZIP must have contained files with names shown on the image below.



To create a BIN or DAT file manually follow the steps below.

Note: we really encourage you to use the **nrf** tool.

## HOW TO CALCULATE CRC -> FIRST GENERATE BIN

To calculate the CRC you need to extract the bytes that will be transferred OTA to the device. The following applications may be found useful:

**hex2bin.exe** – can convert the Intel HEX format to BIN

**crc.exe** – can calculate the CRC from the BIN file

**Sublime Text 3** – can be used to create DAT file or to modify the HEX (see below) – this may also be done by other advanced text editor, e.g. Notepad++.

Each of the following firmware types needs slightly different procedure to generate the BIN file

### 1. Application

This type is the easiest one. If you have the HEX file, just run the command **hex2bin.exe [my\_hex\_file].hex**

### 2. Bootloader

If you perform the same for the bootloader.hex you may notice that the BIN file is much bigger than the HEX. This is because there is a jump to address 0x10001014 at the end of the file. If you open the too-big BIN file you may notice that there are 6 bytes at the beginning followed by a lot of 0xFF and much farther there is the real bootloader content. To fix this problem you have to:

- remove the 2 lines (one with the jump and the following line, shown in bold below) from the HEX file,
- save it,
- create the BIN file (**hex2bin.exe [modified\_bootloader].hex**),
- restore changes in HEX

```
[...]                               # The whole file ...
:10 F7D0 0 [data] 29 # Data record @ 63440
:02 0 4 1000 EA # Extended Linear Address Record @ 0
:04 1014 0 00C00300 15 # Data record @ 4116
:04 0 5 0003C0C1 73 # Start Linear Address Record @ 0
:00 0 1 FF # End Of File record @ 0
```

### 3. Soft Device

A similar operation has to be performed as for the bootloader, but, instead of removing the lines at the end of the file, the MBR (Master Boot Record) needs to be removed for the time of BIN generation.

The MBR section on Soft Device 7.0.0 and 7.1.0 is located from address 0x0000 to 0x1000.

For the Intel HEX documentation check: <http://www.interlog.com/~speff/usefulinfo/Hexfrmt.pdf>. The data address has been marked red on the statement above.

All lines with the address lower than 0x1000 must be temporary deleted. This is all lines starting from the second one (the first one is the jump to address with MSB = 0x0000) until the one with 0x1000 is found.

For the SD 7.1.0 this is up to line 125 (and data address equal 0x07B0).

When removed create the BIN the same way as before: **hex2bin.exe [modified\_sd].hex**

#### 4. Bootloader and Soft device

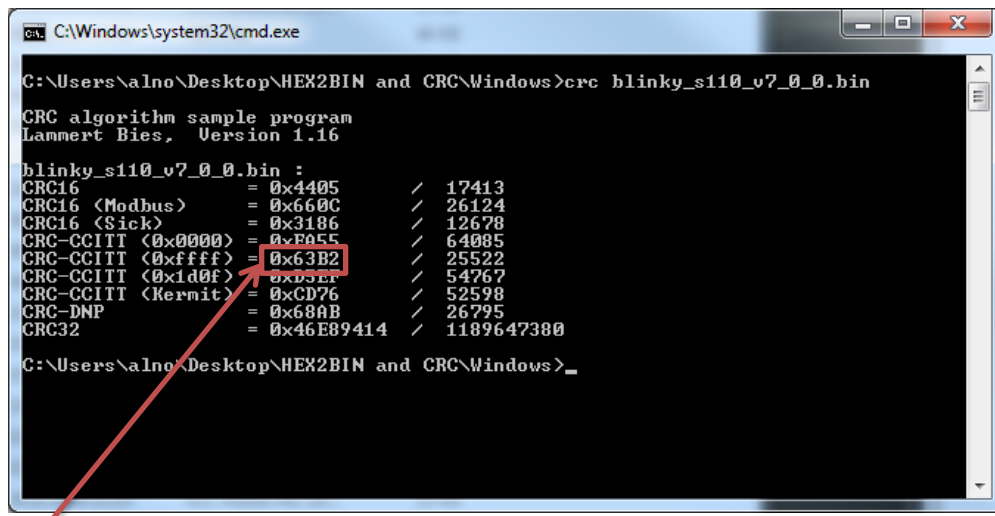
All mobile applications (for Android and iOS) that support DFU allow you to upload the new soft device together with the compatible bootloader. They use ZIP file in which 2 files must be provided, one for the SD and one for the bootloader. However, despite that there are 2 files in the ZIP, they are being sent in one connection and the DFU bootloader checks the CRC for the combined part. You need to create a temporary BIN file with the combined Soft device and bootloader binaries to calculate the CRC that then may be used put to the ZIP file.

To create a combined BIN create first BIN files for both the Soft device and the bootloader and then run the following command: **copy /b [softdevice].bin + [bootloader].bin combined.bin**

#### HOW TO GENERATE CRC -> THEN CALCULATE CRC FROM THE BIN

When you have the BIN file ready you may calculate the CRC file using the **crc.exe** program.

Call: **crc.exe [my\_file].bin**



```
C:\Windows\system32\cmd.exe

C:\Users\alno\Desktop\HEX2BIN and CRC\Windows>crc blinky_s110_v7_0_0.bin

CRC algorithm sample program
Lammert Bies, Version 1.16

blinky_s110_v7_0_0.bin :
CRC16          = 0x4405      / 17413
CRC16 <Modbus>  = 0x660C      / 26124
CRC16 <Sick>    = 0x3186      / 12678
CRC-CCITT (0x0000) = 0xF055    / 64085
CRC-CCITT (0xffff) = 0x63B2    / 25522
CRC-CCITT (0x1d0f) = 0xD5EF    / 54767
CRC-CCITT <Kermit> = 0xCD76    / 52598
CRC-DNP        = 0x680B      / 26795
CRC32          = 0x46E89414   / 1189647380

C:\Users\alno\Desktop\HEX2BIN and CRC\Windows>_
```

The **crc.exe** application creates a number of different CRCs. The bootloader uses the CRC-CCITT with a 0xFFFF seed.

Remember that the CRC must be provided in the reversed order in the init DAT file.