

STM32 Secure Patching Bootloader

Quick Start Guide

By Firmware Modules Inc.

May 13, 2022

V1.1.0

Introduction	1
Integration	2
Integration Step 1 of 4	2
Integration Step 2 of 4	3
Integration Step 3 of 4	7
Integration Step 4 of 4	9
Build Products	9
Notes	10
Release Notes	11

Introduction

The *STM32 Secure Patching Bootloader* is a firmware and software product designed by [Firmware Modules Inc](#) for use with the popular **STM32** family of microcontrollers (MCUs) by [STMicroelectronics](#). As its name suggests, the *STM32 Secure Patching Bootloader* is a bootloader system offering a secure way to update the firmware on an STM32 device using patch files. This bootloader system is designed to be extremely easy to integrate into an existing project: The on-target bootloader binary is pre-built for specific targets and there is nothing to configure. It is a **plug'n'play system** that just works! Unlike other bootloader and firmware update systems, you do not need to be a bootloader expert to get “production level” functionality.

Integrating the [stm32-secure-patching-bootloader](#) into an existing project is a simple **four step process**:

1. Adding bootloader files to your **project repository**.
2. Configuring **STM32CubeIDE**.
3. Adjusting your project's linker script **.ld** file.
4. Adjusting your project's **system_stm32xxxx.c** file.

Each step is described below. We have also created and posted projects you can use as a reference at [stm32-secure-patching-bootloader-demoapp](https://github.com/firmwaremodules/stm32-secure-patching-bootloader-demoapp).

Integration

Integration Step 1 of 4

Add bootloader files to your project repository.

- Create a directory called `Bootloader` in your project root.
- Copy the directories `Include`, `Linker`, `Scripts`, `Tools` into it.
- Copy the `Libs` directory, but only the board subdirectories that you are needing. E.g. copy DISCO-F769I directory into your Libs directory if you are using that board.
- As an alternative, you can add the bootloader files as a git submodule with *git submodule add* <https://github.com/firmwaremodules/stm32-secure-patching-bootloader> *Bootloader*.

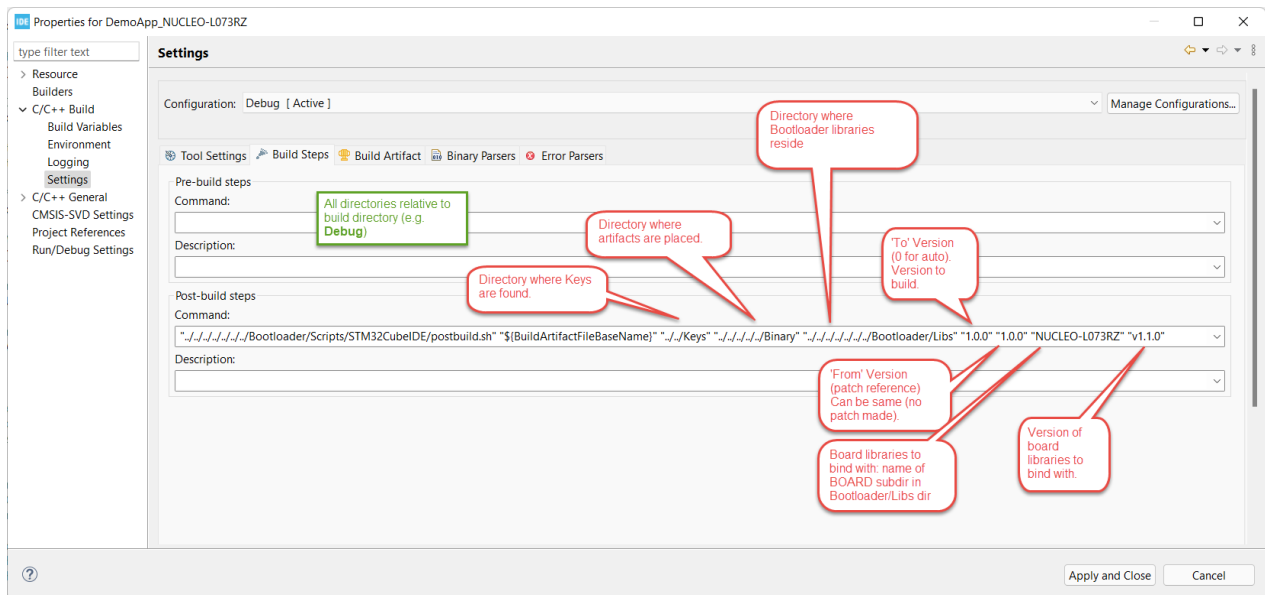
Integration Step 2 of 4

Configure STM32CubeIDE.

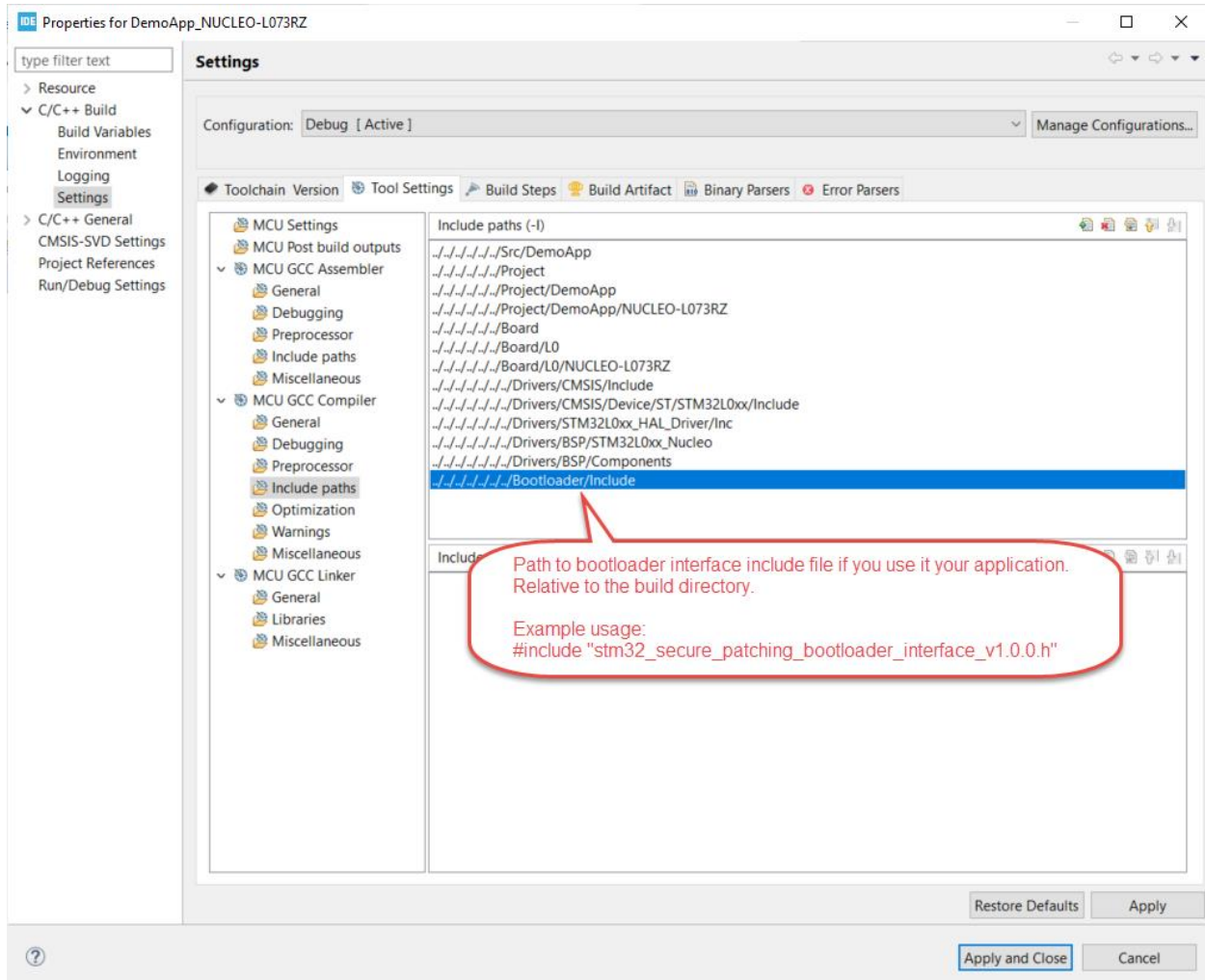
We need to add the **postbuild.sh** script command line, update the include and linker paths, and link with the stm32-secure-patching-bootloader application interface. **C/C++ Build -> Settings -> Build Steps**

Postbuild script command examples for some targets:

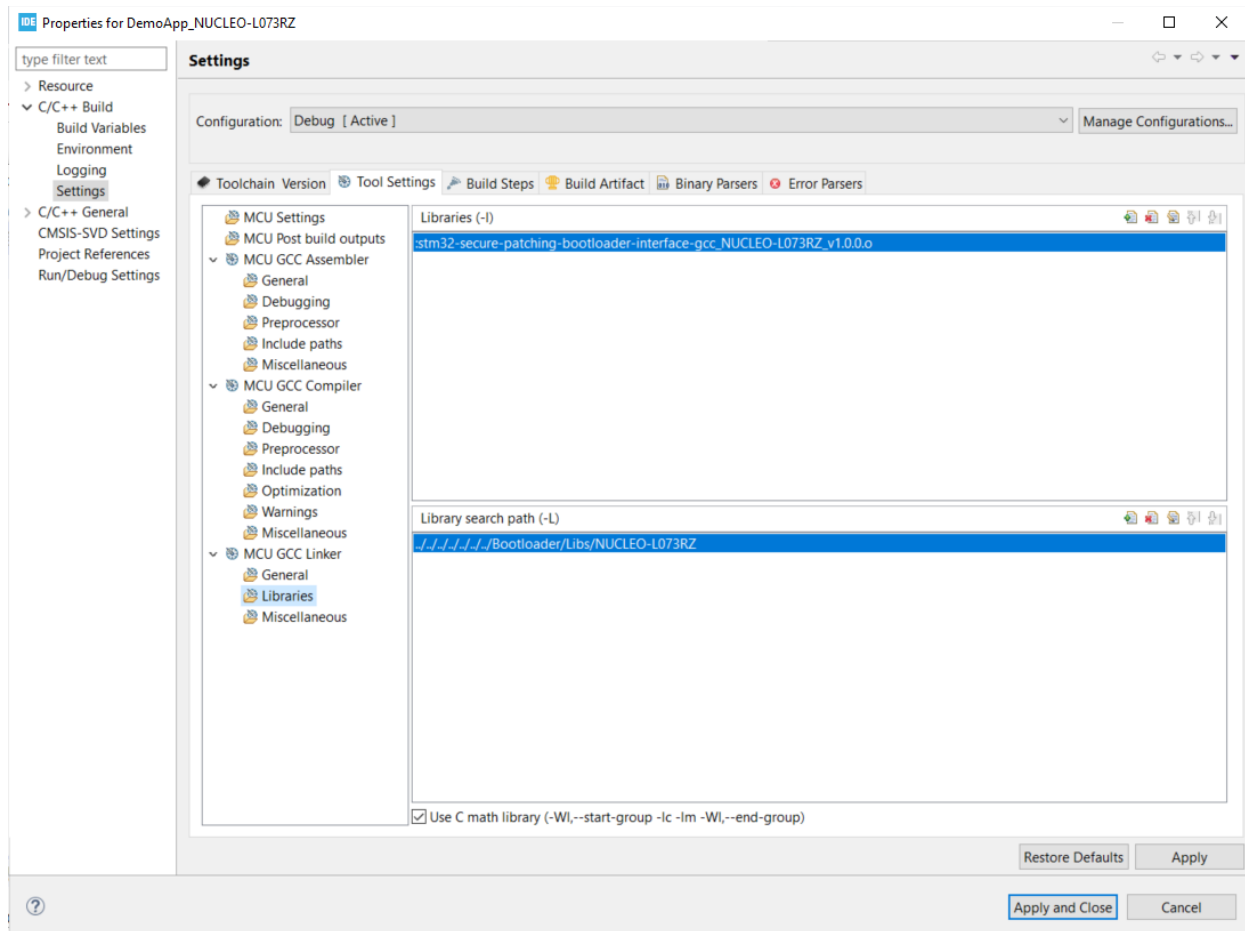
- ```
"../..../..../Bootloader/Scripts/STM32CubeIDE/postbuild.sh"
"${BuildArtifactFileName}" "../..../Keys" "../..../..../Binary" "../..../..../Bootloader
/Libs" "1.0.0" "1.0.0" "NUCLEO-L073RZ" "v1.1.0"
"../..../..../Bootloader/Scripts/STM32CubeIDE/postbuild.sh"
"${BuildArtifactFileName}" "../..../Keys" "../..../..../Binary" "../..../..../Bootloader
/Libs" "1.0.0" "1.0.0" "DISCO-F769I" "v1.1.0"
```



Add path to bootloader include: C/C++ Build -> Settings -> MCU GCC Compiler -> Include paths  
: ../../../../Bootloader/Include



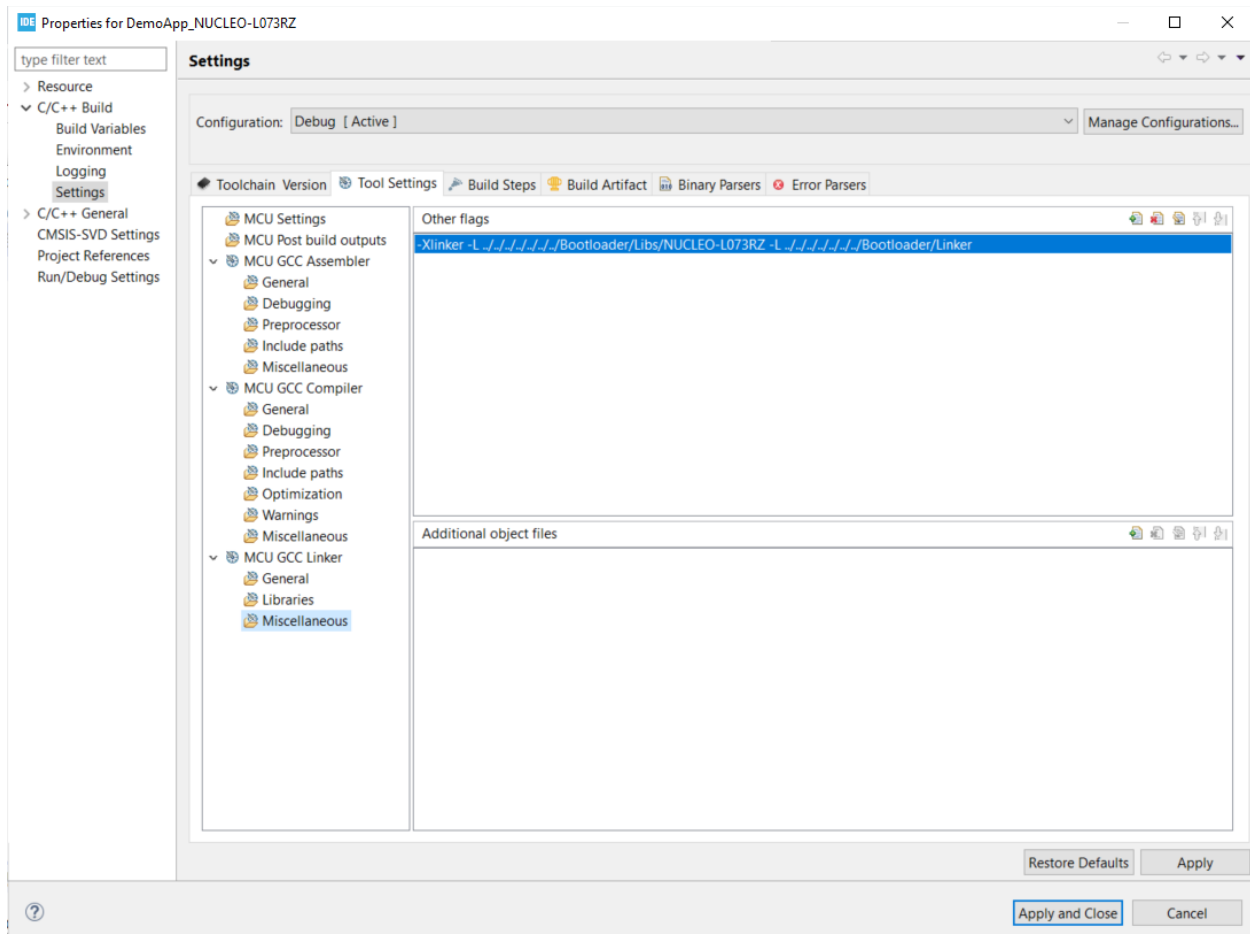
Add paths and links to bootloader library files: **C/C++ Build -> Settings -> MCU GCC Linker -> Libraries** : (Libraries) :stm32-secure-patching-bootloader-interface-gcc\_NUCLEO-L073RZ\_v1.0.0.o  
(Library search paths) ../../../../Bootloader/Libs/NUCLEO-L073RZ



Add paths to bootloader linker files: **C/C++ Build -> Settings -> MCU GCC Linker ->**

**Miscellaneous :** -Xlinker -L ../../../../../../Bootloader/Libs/NUCLEO-L073RZ -L

../../../../../../../../Bootloader/Linker



## Integration Step 3 of 4

Adjust your application's linker script.

At minimum we need to tell the linker where your application's new start offset is in flash. This is defined by the bootloader's linker configuration file **stm32-secure-patching-bootloader-linker-gcc\_<BOARD>\_<version>.ld**.

It defines some symbols that we may need, minimally it is

**STM32\_SECURE\_PATCHING\_BOOTLOADER\_SLOT0\_START**. Then your application's interrupt vector table has to be offset from this by the minimum size of your vector table (to account for the image header), also defined as **VECTOR\_SIZE** in the linker configuration file.

```
/* For this example end of RAM on STM32L073 is 0x20005000 = 20 KB*/
APPLI_region_intvec_start__ = STM32_SECURE_PATCHING_BOOTLOADER_SLOT0_START + VECTOR_SIZE;
APPLI_region_ROM_start = STM32_SECURE_PATCHING_BOOTLOADER_SLOT0_START + VECTOR_SIZE +
VECTOR_SIZE;
APPLI_region_ROM_length = STM32_SECURE_PATCHING_BOOTLOADER_SLOT0_END -
APPLI_region_ROM_start + 1;
APPLI_region_RAM_start = STM32_SECURE_PATCHING_BOOTLOADER_RAM_START;
APPLI_region_RAM_length = 0x20005000 - APPLI_region_RAM_start;

MEMORY
{
 ISR_VECTOR (rx) : ORIGIN = APPLI_region_intvec_start__, LENGTH = VECTOR_SIZE
 APPLI_region_ROM : ORIGIN = APPLI_region_ROM_start, LENGTH = APPLI_region_ROM_length
 APPLI_region_RAM : ORIGIN = APPLI_region_RAM_start, LENGTH = APPLI_region_RAM_length

 /* If your application links with the bootloader interface library, add this section: */
 SE_IF_region_ROM (rx) : ORIGIN = SE_IF_REGION_ROM_START, LENGTH = SE_IF_REGION_ROM_LENGTH
}
...
```

To your SECTIONS, add the following:

```
/* If your application links with the bootloader interface library, add this section
after .isr_vector */

.SE_IF_Code : {
 KEEP(*se_interface_app.o (.text .text*))
} >SE_IF_region_ROM

/* Extra ROM section (last one) to make sure the binary size is a multiple of the AES
block size (16 bytes) */
.align16 :
{
 . = . + 1; /* _edata=. is aligned on 8 bytes so could be aligned on 16
bytes: add 1 byte gap */
 . = ALIGN(16) - 1; /* increment the location counter until next 16 bytes aligned
address (-1 byte) */
}
```

```
 BYTE(0); /* allocate 1 byte (value is 0) to be a multiple of 16 bytes
*/
 } > APPLI_region_ROM
```

A multi-target generic linker script that works with most STM32 targets is provided in the bootloader **Linker** directory. See this [reference project linker script](#).



## Integration Step 4 of 4

### Adjust your application's `system_stm32xxxx.c` file.

This file programs the VTOR register (Vector Table Offset Register) by your application in the early startup phase so that interrupts invoke your application's handlers at the linked offset instead of the bootloader's.

#### `system_stm32xxxx.c` :

```
extern uint32_t APPLI_region_intvec_start__;
#define INTVECT_START ((uint32_t)& APPLI_region_intvec_start__)

void SystemInit(void)
{
 ...

 /* Configure the Vector Table location add offset address -----*/
 SCB->VTOR = INTVECT_START;
}
```

## Build Products

The *stm32-secure-patching-bootloader* **postbuild** process produces three or four artifacts and places them in the specified output directory (typically Binary).

Naming convention: `BOOT_<Your Artifact Name>_<Board Name>_<version>`

1. Combined binaries: `BOOT_DemoApp_NUCLEO-L073RZ_1.0.0.hex`,  
`BOOT_DemoApp_NUCLEO-L073RZ_1.0.0.bin` (.bin produced only when MultiSegment is not enabled)
2. Secured in-field firmware full update file: `DemoApp_NUCLEO-L073RZ_1.0.0.sfb` (full image)
3. Secured in-field firmware patch update file: `DemoApp_NUCLEO-L073RZ_1.0.0_1.1.0.sfbp` (patch - produced only if reference version 1.0.0 exists in Binary\PatchRef)

## Notes

- Patch files .sfbp only work if the source version already exists on the target device. E.g. to update from v1.1.7 to v1.2.0, version v1.1.7 must exist on the device as the active image in SLOT0.
- To update from v1.0.0 to v1.4.0 when only patches of v1.0.0\_v1.1.0, v1.1.0\_v1.2.0, v1.2.0\_v1.3.0, v1.3.0\_v1.4.0 are available, each patch must be applied in series. For each one, firmware is installed by the bootloader through a reboot cycle.
- The build system (STM32CubeIDE) postbuild command line must explicitly set what the reference firmware version is to build a patch from. With this flexibility it is possible to build patches from any version. Commonly this 'from version' is updated from one release to the next to build a series of patchable updates (v1.0.0->v1.1.0, v1.1.0->v1.2.0 etc). Of course, you could create a patch from v1.0.0->v1.2.0 and post this to your update server; if your 'patch picker' algorithm was smart enough it could select the best patch from a list of available patches.
- Full image files .sfb can be used to update to any version and do not depend on what is currently on the device. These could serve as a fall-back in case the hypothetical 'patch picker' implemented in the user application cannot find a suitable patch for download.
- The 'patch picker' is implemented in the USB flash drive updater to select the first patch in the root directory that matches the current version on the device and contains the highest 'to' version. It is able to perform a 'patch chain' update as described above by repeatedly rebooting and updating to the next higher patch version.

## Release Notes

### **V1.1.0 - May 2022**

- Adds new platform support for STM32WLE5 and LORA-E5-DEV board.
- Adds specific support for the B-L072Z-LRWAN1 board.
- Adds README to each library package to describe flash layout and bootloader configuration.
- Changes postbuild command to allow user to specify location of bootloader Libs directory.
- Removes vector offset and multiseg address parameters in the postbuild command script (these are now defined in the bootloader library package artifacts).

### **V1.0.0 - Dec 2021**

- Initial Release