# UMxxxx



**Getting Started with STM32CubeF3 firmware package for STM32F3xx series**

# Introduction

The STM32CubeTM initiative was originated by STMicroelectronics to ease developers’ life by reducing development efforts, time and cost. STM32CubeTM covers the STM32 portfolio.

STM32CubeVersion 1.x includes:

* The STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
* A comprehensive embedded software platform, delivered per series (such as STM32CubeF3 for STM32F3 series)
  + The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  + A consistent set of middleware components such as RTOS, USB, STMTouch, FatFS and Graphics
  + All embedded software utilities coming with a full set of examples.

This user manual describes how to get started with the STM32CubeF3 firmware package.

Chapter 1 describes the main features of STM32CubeF3 firmware, part of the

STM32Cube™ initiative.

Chapter 2 and Chapter 3 provide an overview of the STM32CubeF3 architecture and

firmware package structure.

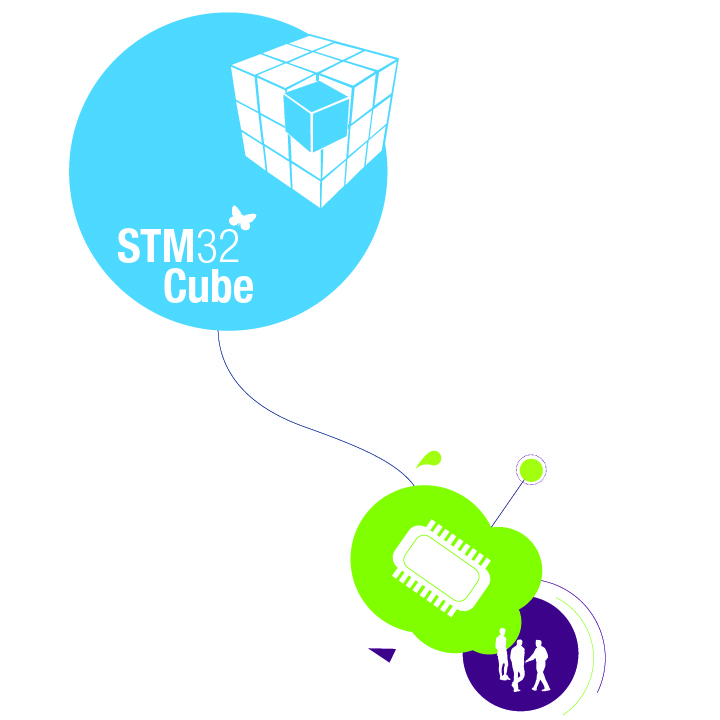


Table of contents

[UMxxxx 1](#_Toc387237371)

[Introduction 1](#_Toc387237372)

[1. STM32CubeF3 main features 5](#_Toc387237373)

[2. STM32CubeF3 architecture overview 6](#_Toc387237374)

[3. STM32CubeF3 firmware package overview 9](#_Toc387237375)

[3.1. Supported STM32F3 devices and hardware 9](#_Toc387237376)

[3.2. Firmware package overview 11](#_Toc387237377)

[4. Getting started 14](#_Toc387237378)

[4.1. Running your first example 14](#_Toc387237379)

[4.2. How to develop your own application 15](#_Toc387237380)

[4.3. Using STM32CubeMX to generate the initialization C code 17](#_Toc387237381)

[4.4. Getting STM32CubeF3 release updates 17](#_Toc387237382)

[4.4.1. Installing and running the STM32CubeUpdater program 17](#_Toc387237383)

[5. FAQ 18](#_Toc387237384)

[Revision history 20](#_Toc387237385)

**List of figures**

[Figure 1. STM32CubeTM firmware components 5](#_Toc387237391)

[Figure 2 STM32CubeF3 firmware architecture 6](#_Toc387237392)

[Figure 3. STM32CubeF3 firmware package structure 11](#_Toc387237393)

[Figure 4. STM32CubeF3 examples overview 12](#_Toc387237394)

**List of tables**

**No table of figures entries found.**

# STM32CubeF3 main features

STM32CubeF3 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32F3 microcontrollers. In line with the STM32CubeTM initiative, this set of components is highly portable, not only within STM32F3 series but also to other STM32 series.

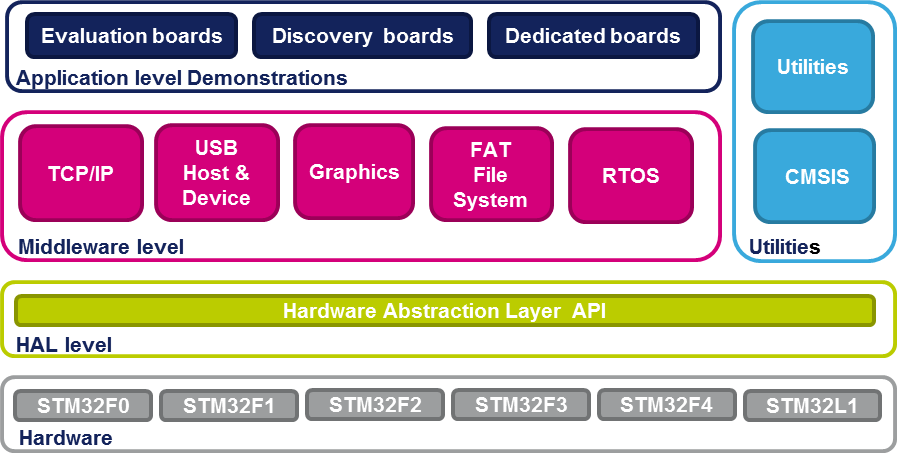
STM32CubeF3 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for user convenience.

STM32CubeF3 package also contains a set of middleware components with the corresponding examples. They come in very permissive license terms:

* Full USB Device stack supporting many classes:
  + Device classes: HID, MSC, CDC, DFU
* CMSIS-RTOS implementation with FreeRTOS open source solution
* FAT File system based on open source FatFS solution
* STMTouch touch sensing library solution
* STemWin, a professional graphical stack solution available in binary format and based on ST partner solution SEGGER emWin

Several applications and demonstrations implementing all these middleware components are also provided in the STM32CubeF3 package.

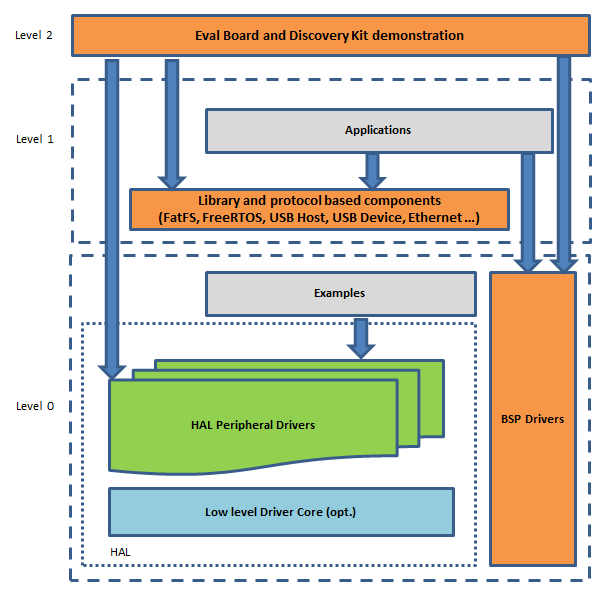
Figure . STM32CubeTM firmware components



# STM32CubeF3 architecture overview

The STM32Cube firmware solution is built around three independent levels that can easily interact with each other’s as described in the figure 2 below:

Figure 2 STM32CubeF3 firmware architecture



**Level 0:** This level is divided into three sub-layers:

* **Board Support Package (BSP):** this layer offers a set of APIs relative to the hardware components in the hardware boards (LCD drivers, MicroSD, etc…) and composed of two parts:
  + Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provide specific APIs to the BSP driver external components and could be portable on any other board.
  + BSP driver: it permits to link the component driver to a specific board and provides a set of friendly used APIs. The APIs naming rule is BSP\_FUNCT\_Action(): ex. BSP\_LED\_Init(),BSP\_LED\_On()

It’s based on modular architecture allowing an easy porting on any hardware by just implementing the low level routines.

* **Hardware Abstraction Layer (HAL):** this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi instance and functionalities oriented APIs which permit to offload the user application implementation by providing ready to use process. As example, for the communication peripherals (I2S, UART…) it provides APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and manage communication errors that may raise during communication. The HAL Drivers APIs are split in two categories, generic APIs which provides common and generic functions to all the STM32 series and extension APIs which provides specific and customized functions for a specific family or a specific part number.
* **Basic peripheral usage examples**: this layer encloses the examples build over the STM32 peripheral using only the HAL and BSP resources.

**Level 1:** This level is divided into two sub-layers:

* **Middleware components**: set of Libraries covering USB Device library, STMTouch touch sensing library, graphical STemWin library, FreeRTOS and FatFS. Horizontal interactions between the components of this layer is done directly by calling the feature APIs while the vertical interaction with the low level drivers is done through specific callbacks and static macros implemented in the library system call interface. As example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.

The main features of each Middleware component are as follows:

USB Device Library

– Several USB classes supported (Mass-Storage, HID, CDC, DFU, AUDIO, MTP)

– Supports multi packet transfer features: allows sending big amounts of data without splitting them into max packet size transfers.

– Uses configuration files to change the core and the library configuration without changing the library code (Read Only).

– RTOS and Standalone operation

– The link with low-level driver is done through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level

drivers.

FreeRTOS

* Open source standard
* CMSIS compatibility layer
* Tickless operation during low-power mode
* Integration with all STM32Cube Middleware modules

FAT File system

* FATFS FAT open source library
* Long file name support
* Dynamic multi-drive support
* RTOS and standalone operation
* Examples with microSD.

STM32 Touch Sensing Library

* Robust STMTouch capacitive touch sensing solution supporting proximity, touchkey, linear and rotary touch sensor using a proven surface charge transfer acquisition principle.

STemWin Library

* Graphical library supporting LCD provided as part as the STM32CubeF3 firmware package.
* **Examples based on the Middleware components**: each Middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several Middleware components are provided as well.

**Level 2:** This level is composed of a single layer which is global real-time and graphical demonstration based on the Middleware service layer, the low level abstraction layer and the basic peripheral usage applications for board based functionalities.

# STM32CubeF3 firmware package overview

# Supported STM32F3 devices and hardware

STM32CubeTM is offering highly portable Hardware Abstraction Layer (HAL) built around a generic architecture and allows the build-upon layers, like the middleware layer, to implement its functions without knowing, in-depth, the MCU used. This improves the library code reusability and guarantees an easy portability on other devices.

And thanks to its layered architecture the STM32CubeF3 offers full support of all STM32F3 Series, user has only to define the right macro in stm32f3xx.h.

Table 1 below provides the macro to be defined depending on the used STM32F3 device (this macro should be defined as well in the compiler preprocessor).

**Table 1. Macros for STM32F3 series**

|  |  |
| --- | --- |
| Macro defined in stm32f3xx.h | STM32F3 devices |
| STM32F301x8 | STM32F301K6, STM32F301C6, STM32F301R6,  STM32F301K8, STM32F301C8 and STM32F301R8 |
| STM32F302x8 | STM32F302K6, STM32F302C6, STM32F302R6,  STM32F302K8, STM32F302C8 and STM32F302R8 |
| STM32F302xC | STM32F302CB, STM32F302RB, STM32F302VB,  STM32F302CC, STM32F302RC and STM32F302VC |
| STM32F303x8 | STM32F303K6, STM32F303C6, STM32F303R6,  STM32F303K8, STM32F303C8 and STM32F303R8 |
| STM32F303xC | STM32F303CB, STM32F303RB, STM32F303VB,  STM32F303CC, STM32F303RC and STM32F303VC |
| STM32F373xC | STM32F373C8, STM32F373R8, STM32F373V8,  STM32F373CB, STM32F373RB, STM32F373VB,  STM32F373CC, STM32F373RC and STM32F373VC |
| STM32F334x8 | STM32F334K4, STM32F334C4, STM32F334R4,  STM32F334K6, STM32F334C6, STM32F334R6,  STM32F334K8, STM32F334C8 and STM32F334R8 |
| STM32F318xx | STM32F318K8 and STM32F318C8 |
| STM32F328xx | STM32F328C8 and STM32F328R8 |
| STM32F358xx | STM32F358CC, STM32F358RC and STM32F358VC |
| STM32F378xx | STM32F378CC, STM32F378RC and STM32F378VC |

STM32CubeF3 features a rich set of examples and applications at all levels making it easy to understand and use any HAL driver and/or Middleware components. These examples are running on STMicroelectronics boards as listed in Table 2 below:

**Table 2. Boards for STM32F3 series**

|  |  |
| --- | --- |
| Board | STM32F3 devices supported |
| STM32303C-EVAL | STM32F303xC |
| STM32373C-EVAL | STM32F373xC |
| STM32F3-Discovery | STM32F303xC |
| STM32F3348-Discovery | STM32F334x8 |
| NUCLEO-F302R8 | STM32F302x8 |
| NUCLEO-F334R8 | STM32F334x8 |

As for all other STM32 Nucleo boards, the NUCLEO-L302R8 and NUCLEO-L334R8 feature a reduced set of hardware components (one user Key button and one user LED). And in order to enrich the middleware support offer for in STM32CubeF3 firmware package, an LCD display Adafruit Arduino shield was chosen, which embeds in addition to the LCD a μSD connector and Joystick.

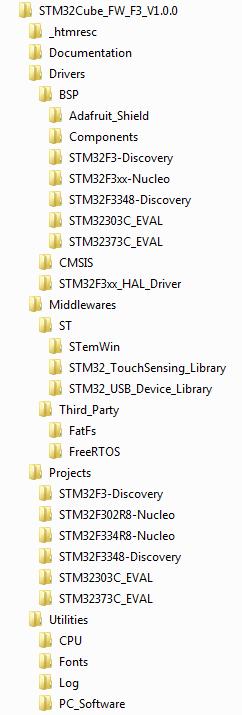
In the BSP component the dedicated drivers, for that Arduino shield are available and their use is illustrated through either the provided BSP example or in the Demonstration firmware, without forgetting the FatFS middleware application.

The STM32CubeF3 firmware is able to run on any compatible hardware. That means user can simply update the BSP drivers to port the provided examples on his own board, if this later has the same hardware functionalities (LED, LCD Display, Buttons...etc.).

# Firmware package overview

The STM32CubeF3 firmware solution is provided in one single zip package having the structure shown in Figure 2 below

Figure . STM32CubeF3 firmware package structure



User modifiable files

Open sources Middleware stacks

USB Device Library offering the following classes: HID, MSC, CDC and DFU

TouchSensing Library

STM32F3xx HAL drivers for all peripherals

Drivers of external components

Miscellaneous utilities

Set of Examples, Applications and Demonstrations organized by board and provided with preconfigured projects

Contains STM32F3xx CMSIS files that defines Peripheral's registers declarations, bits definition and the address mapping

STemWin professional stack coming from SEGGER and available in binary form

BSP drivers for the supported boards:

* Evaluation board
* Discovery kit
* Nucleo kit

Library files, to not be modified by user

Figure . STM32CubeF3 examples overview

|  |  |
| --- | --- |
| For each board, a set of examples are provided with preconfigured projects for EWARM, MDK-ARM and TrueSTUDIO toolchains.  Figure 4 shows the projects structure for the STM32303C-EVAL board.  The examples are classified depending on the STM32Cube level they apply to, and are named as below:   * examples in level 0 are called **Examples**, that uses HAL drivers without any Middleware component * examples in level 1 are called **Applications**, that provides typical use cases of each Middleware component   The **Template** project available in the Template directory is provided to build quickly any firmware application on a given board.  All examples have the same structure,   * **\Inc** folder that contains all header files * **\Src** folder for the sources code * **\EWARM**, **\MDK-ARM** and **\TrueSTUDIO** folders contain the preconfigured project for each toolchain. * **readme.txt** describing the example behavior and needed environment to make it working |  |

Table 2 below provides the number of projects available for each board.

**Table 2. Number of examples available for each board**

|  |  |  |  |
| --- | --- | --- | --- |
| Board | Examples | Applications | Demonstration |
| STM32303C\_EVAL | 54 | 15 | N/A |
| STM32373C\_EVAL | 40 | 15 | N/A |
| STM32F3-Discovery | 38 | 3 | 1 |
| STM32F302R8-Nucleo | 27 | 3 | 1 |
| STM32F3348-Discovery | 16 | 1 | 1 |
| STM32F334R8-Nucleo | 2 | 1 | 1 |

# Getting started

# Running your first example

This section explains how simple is to run a first example within STM32CubeF3, using as illustration the generation of a simple LED toggle running on STM32F302R8 Nucleo board:

1. Download the STM32CubeF3 firmware package. Unzip it into a directory of your choice. Make sure not to modify the package structure shown in Figure 4 above). Note that it is also recommended to copy the package at a location close to your root volume (i.e. C\Eval or G:\Tests) because some IDEs encounter problems when Path length is too high.
2. Browse to \Projects\STM32F302R8-Nucleo\Examples
3. Open \GPIO, then \GPIO\_EXTI folder
4. Open the project with your preferred toolchain (\*)
5. Rebuild all files and load your image into target memory
6. Run the example: each time you press the USER push button, the LED2 toggles (for more details, refer to the example readme file).

(\*) Below a quick overview on how to open, build and run an example with the supported toolchains:

* EWARM
  + Under the example folder, open \EWARM subfolder
  + Launch the Project.eww workspace (\*\*)
  + Rebuild all files: **Project->Rebuild all**
  + Load project image: **Project->Debug**
  + Run program: **Debug->Go(F5)**
* MDK-ARM
  + Under the example folder, open \MDK-ARM subfolder
  + Launch the Project.uvproj workspace (\*\*)
  + Rebuild all files: **Project->Rebuild all target files**
  + Load project image: **Debug->Start/Stop Debug Session**
  + Run program: **Debug->Run (F5)**
* TrueSTUDO
  + Open the TrueSTUDIO toolchain
  + Click on **File->Switch Workspace->Other** and browse to TrueSTUDIO workspace directory
  + Click on **File->Import**, select **General->'Existing Projects into Workspace'** and then click "**Next**".
  + Browse to the TrueSTUDIO workspace directory, select the project
  + Rebuild all project files: Select the project in the "**Project explorer**" window then click on **Project->build project** menu.
  + Run program: **Run->Debug (F11)**

(\*\*) The workspace name may changes from one example to another

# How to develop your own application

This section describes the needed steps to create your own application using STM32CubeF3.

1. **Create your project:** to create a new project you can either start from the **Template** project provided for each board under \Projects\<STM32xxx\_yyy>\Templates or from any available project under \Projects\<STM32xxy\_yyy>\Examples or \Projects\<STM32xx\_yyy>\Applications (<STM32xxx\_yyy> refers to the board name, ex. STM32303C\_EVAL).

The Template project is providing empty main loop function, however it’s a good starting point to get familiar with project settings for STM32CubeF3. Below its characteristics:

* 1. It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board
  2. It contains the include paths for all the firmware components
  3. It defines the STM32F3 device supported, allowing to configure the CMSIS and HAL drivers accordingly
  4. It provides ready to use user files preconfigured as below
     1. HAL is initialized with default time base with ARM Core SysTick
     2. SysTick ISR implemented for HAL\_Delay() purpose
     3. System clock configured with the minimum frequency of the device (HSI) for an optimum power consumption.

*Note: If you copy an existing project to another location, then you need to update the include paths.*

1. **Add the necessary Middleware to your project (optional):** available Middleware stacks are: USB Device Library, STemWin, Touch Sensing Library, FreeRTOS and FatFS. To know which source files you need to add in the project files list; refer to the documentation provided for each Middleware, you may have a look also to the Applications available under \Projects\STM32xxx\_yyy\Applications\<MW\_Stack> (<MW\_Stack> refers to the Middleware stack, ex USB\_Device) to have better idea on the sources files to be added and the include paths.
2. **Configure the firmware components:** the HAL and Middleware components offer a set of build time configuration options using macros “#define” declared in a header file. A template configuration file is provided within each component, it has to be copied to the project folder (usually the configuration file is named xxx\_conf\_template.h, the word “\_template” need to be removed when copying it to the project folder). The configuration file provides enough information to know the impact of each configuration option; more detailed information is available in the documentation provided for each component.
3. **Start the HAL Library**: after jumping to the main program, the application code need to call *HAL\_Init()* API to initialize the HAL Library, which do the following:
4. configure the Flash prefetch and SysTick interrupt priority (configured by user through macros defined in stm32f3xx\_hal\_conf.h)
5. configure the Systick to generate an interrupt each 1 msec at the SysTick interrupt priority TICK\_INT\_PRIO defined in stm32f3xx\_hal\_conf.h, which is clocked by the HSI (at this stage, the clock is not yet configured and thus the system is running from the internal HSI at 8 MHz)
6. Set NVIC Group Priority to 4
7. Calls HAL\_MspInit() callback function defined in user file stm32f3xx\_hal\_msp.c to do global low level hardware initializations
8. **Configure the system clock:** the system clock configuration is done by calling these two APIs
9. HAL\_RCC\_OscConfig(): configures the internal and/or external oscillators, PLL source and factors. User may select to configure one oscillator or all oscillators, in addition PLL configuration may be skipped if there is no need to run the system at high frequency
10. HAL\_RCC\_ClockConfig(): configures the system clock source, Flash latency and AHB and APB prescaler
11. **Peripheral initialization**
12. Start by writing the peripheral HAL\_PPP\_MspInit function. For this function, please proceed as follows
    1. Enable the peripheral clock.
    2. Configure the peripheral GPIOs.
    3. Configure DMA channel and enable DMA interrupt (if needed).
    4. Enable peripheral interrupt (if needed).
13. Edit the stm32xxx\_it.c to call required interrupt handlers (peripheral and DMA), if needed.
14. Write process complete callback functions if you plan to use peripheral interrupt or DMA.
15. In your main.c file, initialize the peripheral handle structure then call the function

HAL\_PPP\_Init() to initialize your peripheral.

1. **Developing your application process:** at this stage, your system is ready and you can start developing your application code.
2. The HAL provides intuitive and ready to use APIs to configure the peripheral, and supports polling, IT and DMA programming model, to accommodate any application requirements. For more details on how to use each peripheral, refer to the rich examples set provided.
3. If your application has some real time constraints, you can found a large set of examples showing how to use FreeRTOS and its integration with all Middleware stacks provided within STM32CubeF3, it can be a good starting point for your development.

**Caution:** In the default HAL implementation, Systick timer is the source of time base. It is used to generate interrupts at regular time intervals. Take care if HAL\_Delay() is called from peripheral ISR process. The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise, the caller ISR process is blocked.. Functions affecting time base configurations are declared as \_\_weak to make override possible in case of other implementations in user file (using a general purpose timer for example or other time source), for more details please refer to HAL\_TimeBase example.

# Using STM32CubeMX to generate the initialization C code

An alternative to steps 1 to 6 described in Section 4.2consists in using the STM32CubeMX

tool to generate code for the initialization of the system, the peripherals and middleware

(steps 1 to 6 above) through a step-by-step process:

* Select the STMicroelectronics STM32 microcontroller that matches the required set of

peripherals.

* Configure each required embedded software thanks to a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and the utility performing MCU

peripheral configuration (for example GPIO, USART) and middleware stacks (for

example USB).

* Generate the initialization C code based on the configuration selected. This code is

ready to use within several development environments. The user code is kept at the

next code generation.

For more information, please refer to UM1718.

# Getting STM32CubeF3 release updates

The STM32CubeF3 firmware package comes with an updater utility: STM32CubeUpdater, also available as a menu within STM32CubeMX code generation tool.

The updater solution detects new firmware releases and patches available from st.com and proposes to download them to the user’s computer.

# Installing and running the STM32CubeUpdater program

* Double-click on the SetupSTM32CubeUpdater.exefile to launch the installation.
* Accept the license terms and follow the different installation steps.

Upon successful installation, STM32CubeUpdater becomes available as an STMicroelectronics program under Program Files and is automatically launched.

The STM32CubeUpdater icon appears in the system tray:

Right-click the updater icon and select **Updater Settings** to configure the Updater connection and whether to perform manual or automatic checks (see STM32CubeMX User guide - UM1718 section 3 - for more details on Updater configuration).

# FAQ

**What is the license scheme for the STM32CubeF3 firmware?**

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license. The Middleware stacks made by ST (USB Hostand Device Libraries, STemWin) come with a licensing model allowing easy reuse, provided it runs on an ST device.

The Middleware based on well-known open-source solutions (FreeRTOS and FatFs) have user-friendly license terms. For more details, refer to the license agreement of each Middleware.

**What boards are supported by the STM32CubeF3 firmware package?**

The STM32CubeF3 firmware package provides BSP drivers and ready-to-use examples for the following STM32F3 boards: STM32303C-EVAL, STM32373C-EVAL, STM32F3-Discovery, STM32F3348-Discovery, NUCLEO-F302R8 and NUCLEO-F334R8.

**Is there any link with Standard Peripheral Libraries?**

The STM32Cube HAL Layer is the replacement of the Standard Peripheral Library.

The HAL APIs offer a higher abstraction level compared to the standard peripheral APIs.

HAL focuses on peripheral common functionalities rather than hardware. The higher abstraction level allows to define a set of user friendly APIs that can be easily ported from one product to another.

Customers currently using Standard Peripheral Libraries will be helped through Migration guides. Existing Standard Peripheral Libraries will be supported, but not recommended for new designs.

**Does the HAL take benefit from interrupts or DMA? How can this be controlled?**

Yes. The HAL supports three API programming models: polling, interrupt and DMA (with or without interrupt generation).

**Are any examples provided with the ready-to-use toolset projects?**

Yes. STM32CubeF3 provides a rich set of examples and applications (around 70 for STM32303C-EVAL). They come with the preconfigured project of several toolsets: IAR, Keil and GCC.

**How are the product/peripheral specific features managed?**

The HAL offers extended APIs, i.e. specific functions as add-ons to the common API to support features available on some products/lines only.

**How can STM32CubeMX generate code based on embedded software?**

STM32CubeMX has a built-in knowledge of STM32 microcontrollers, including their peripherals and software. This enables the tool to provide a graphical representation to the user and generate \*.h/\*.c files based on user configuration.

**How to get regular updates on the latest STM32CubeF3 firmware releases?**

The STM32CubeF3 firmware package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new firmware package updates (new releases or/and patches).

STM32CubeUpdater is integrated as well within the STM32CubeMX tool. When using this tool for STM32F3 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeF3 firmware package updates.

For more details, refer to Section 4.4.

# 

# Revision history

| Revision | Date of modifications | Author | Description of modifications |
| --- | --- | --- | --- |
| 0.1 | 05-06-2014 | MCD Application | Initial draft version |
| 0.2 | 05-06-2014 | MCD Application | 2nd draft version aver RHO comments:   * changed Figures 1 and 2 * added FAQ |

**Please Read Carefully:**

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

**UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZE REPRESENTATIVE OF ST, ST PRODUCTS ARE NOT DESIGNED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS, WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.**

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2006 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan -

Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com