UM1730

User manual

Getting started with STM32CubeF4 for STM32F4 series MCUs

Introduction

STM32Cube is an STMicroelectronics original initiative to improve designer productivity significantly by reducing development effort, time, and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:

- A set of user-friendly software development tools to cover project development from conception to realization, among which are:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeCLT, an all-in-one command-line development toolset with code compilation, board programming, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor (STM32CubeMonitor, STM32CubeMonPwr, STM32CubeMonRF, STM32CubeMonUCPD), powerful monitoring tools to fine-tune the behavior and performance of STM32 applications in real time
- STM32Cube MCU and MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeF4 for the STM32F4 series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
  - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over hardware
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, graphics, and FAT file system
  - All embedded software utilities with full sets of peripheral and applicative examples
- STM32Cube Expansion Packages, which contain embedded software components that complement the functionalities of the STM32Cube MCU and MPU Packages with:
  - Middleware extensions and applicative layers
  - Examples running on some specific STMicroelectronics development boards

This user manual describes how to get started with the STM32CubeF4 MCU Package.

Section 2 describes the main features of STM32CubeF4 MCU package.

Section 3 and Section 4 provide an overview of the STM32CubeF4 architecture and MCU package structure.

1 General information

The STM32CubeF4 MCU package runs on STM32F4 series series 32-bit microcontrollers based on the Arm® Cortex®-M4 processor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 STM32CubeF4 main features

The STM32CubeF4 MCU package runs on STM32 microcontrollers based on the Arm® Cortex®-M4 processor.

STM32CubeF4 gathers all generic embedded software components required to develop an application on STM32F4 microcontrollers in a single package. In line with the STMCube initiative, this set of components is highly portable, not only within the STM32F4 series but also on other STM32 series.

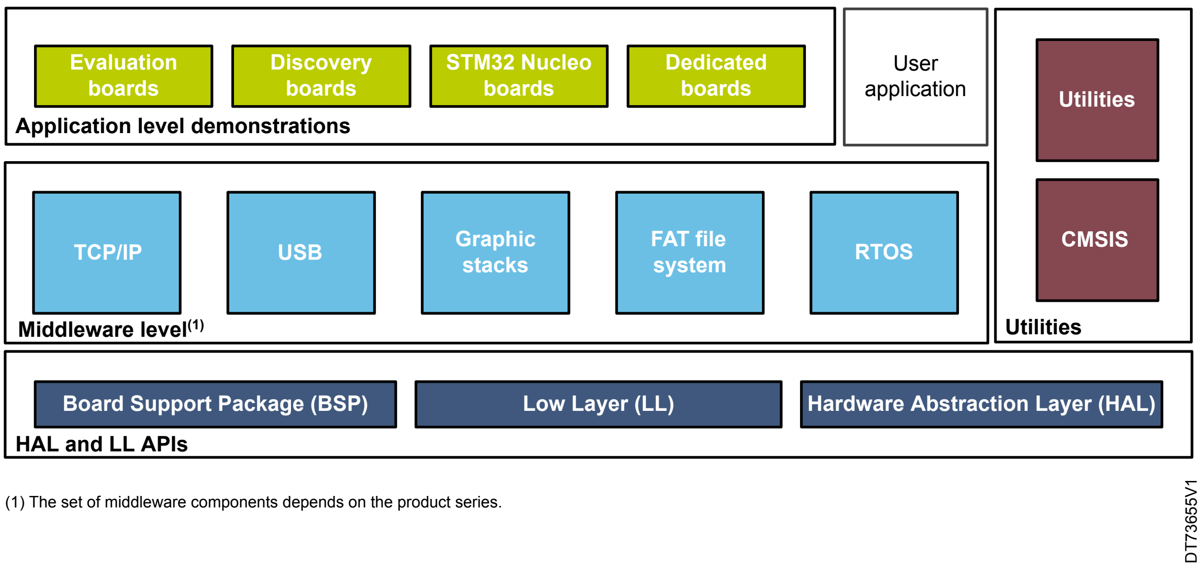
STM32CubeF4 is fully compatible with the STM32CubeMX code generator for generating initialization code. The package includes low-layer (LL) and hardware abstraction layer (HAL) APIs that cover the microcontroller hardware, with an extensive set of examples running on STMicroelectronics boards. The HAL and LL APIs are available in an open-source BSD license for user convenience.

The STM32CubeF4 MCU package also contains a set of middleware components with the corresponding examples. They come with very permissive license terms:

- A full USB host and device stack supporting many classes:
  - Host classes: HID, MSC, CDC, Audio, MTP.
  - Device classes: HID, MSC, CDC, Audio, DFU.
- Graphics:
  - STemWin, a professional graphical stack solution available in binary format and based on the emWin solution from the STMicroelectronics' partner Segger.
  - ST-TouchGFX, an STMicroelectronics professional graphical stack solution available in binary format.
  - LibJPEG, an open-source implementation on STM32 for JPEG image encoding and decoding.
- CMSIS-RTOS implementation with a FreeRTOS™ open-source solution.
- FAT file system based on an open-source FatFS solution.
- TCP/IP stack based on an open-source LwIP solution.
- SSL/TLS secure layer based on the open-source mbedTLS.

A demonstration implementing all these middleware components is also provided in the STM32CubeF4 package.

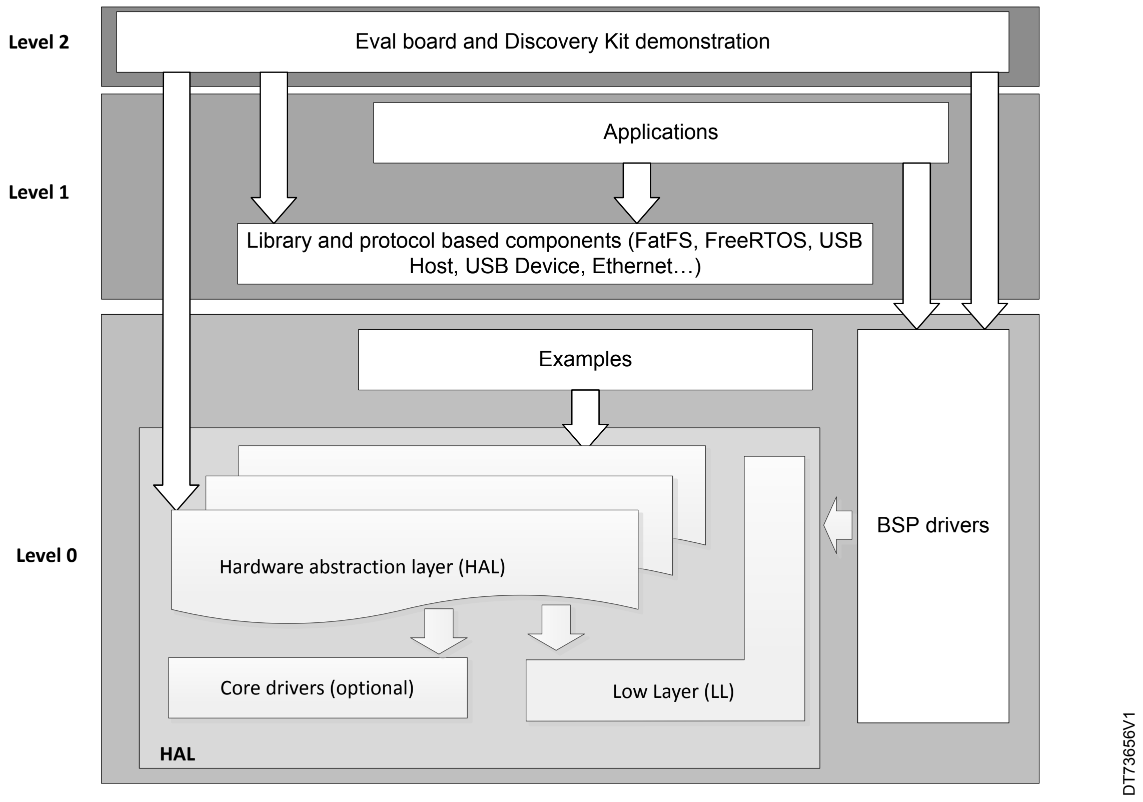
Figure 1. **STM32CubeF4 firmware components**



3 STM32CubeF4 architecture overview

The STM32CubeF4 firmware solution is built around three independent levels that can easily interact with each other as described in the following figure:

Figure 2. STM32CubeF4 firmware architecture


Level 2
Eval board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components (FatFS, FreeRTOS, USB Host, USB Device, Ethernet…)
Level 0
Examples
Hardware abstraction layer (HAL)
Core drivers (optional)
Low Layer (LL)
HAL
BSP drivers
DT73656V1

3.1 Level 0

This level is divided into three sublayers:

- Board support package (BSP).
- Hardware abstraction layer (HAL).
- Low layer (LL).

Board support package (BSP)

This layer offers a set of APIs related to the hardware components on the hardware boards (audio codec, I/O expander, touchscreen, SRAM driver, LCD drivers, and others) and is composed of two parts:

- Component: the driver related to the external device on the board and not related to the STM32. The component driver provides specific APIs to the BSP driver external components and can be ported to any board
- BSP driver: enables the component driver to be linked to a specific board and provides a set of user-friendly APIs. The API naming rule is *BSP_FUNCT_Action()*, for example, *BSP_LED_Init(),BSP_LED_On()*.

It is based on a modular architecture that allows it to be ported easily to 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 function-oriented APIs that allow the offloading of the user application implementation by providing ready-to-use processes. For example, for the communication peripherals (such as I^2S and 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 arise during communication. The HAL driver APIs are split into two categories:

- Generic APIs that provide common and generic functions to all the STM32 series.
- Extension APIs that provide specific and customized functions for a specific family or part number.

Basic peripheral usage examples: this layer contains examples of basic operations of the STM32F4 peripherals using only the HAL and/or the low-layer driver APIs, as well as the BSP resources.

Low layer (LL)

This layer provides low-level APIs at register level, with better optimization but less portability. They require a deep knowledge of the MCU and its peripherals. The LL drivers form a light-weight expert-oriented layer that is closer to the hardware than the HAL. As opposed to the HAL, LL APIs are not provided for peripherals where optimized access is not key, or for peripherals requiring heavy software configuration and/or complex upper-level stack (such as FSMC, USB, or SDMMC).

Features of the LL drivers:

- A set of functions to initialize the main features of a peripheral according to the parameters specified in data structures.
- A set of functions used to fill initialization data structures with the reset values corresponding to each field.
- A function for the deinitialization of a peripheral (registers restored to their default values).
- A set of inline functions for direct and atomic register access.
- Full independence of HAL and capability to be used in standalone mode (without HAL drivers).
- Full coverage of the features of a peripheral.

3.2 Level 1

This level is divided into two sublayers:

- Middleware components.
- Examples based on the middleware components.

Middleware components

The middleware components are a set of libraries covering USB host and device libraries, STemWin, ST-TouchGFX, LibJPEG, FreeRTOS™, FatFS, LwIP, and mbedTLS. Horizontal interactions between the components of this layer are 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. For example, FatFS implements the disk I/O driver to access the microSD™ drive or the USB mass storage class.

The main features of each middleware component are as follows:

- USB host and device libraries:
  - Several USB classes are supported (Mass-Storage, HID, CDC, DFU, AUDIO, MTP).
  - Support multipacket transfer features: allow sending large amounts of data without splitting them into maximum packet size transfers.
  - Use configuration files to change the core and the library configuration without changing the library code (read-only).
  - Include 32-bit aligned data structures to handle DMA-based transfer in high-speed modes.
  - Support multi-USB OTG core instances from user level through configuration file (allow operations with more than one USB host/device peripheral).
  - RTOS and standalone operation.
  - The link with the 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.

- STemWin graphical stack:
  - Professional-grade solution for GUI development based on Segger's emWin solution.
  - Optimized display drivers.
  - Software tools for code generation and bitmap editing (for example, STemWin Builder).
- ST-TouchGFX graphical stack:
  - ST-TouchGFX, a modern C++ framework for high-end graphics on STM32 microcontrollers.
  - Screen transition effects and easing equations for animations.
  - Image dithering, alpha blending, 2D/3D image rotation, and scaling (no extra memory required).
  - Fast occlusion culling, avoiding the display of hidden pixels in widgets.
  - Rasterization of lines, circles, and custom shapes, with antialiasing.
  - Direct rendering from memory-mapped flash, direct frame buffer manipulation for custom widgets, and extensive use of Chrom-ART hardware acceleration for rendering.
  - TrueType and OpenType font support, 1-, 2-, 4-, and 8-bit antialiased text rendering, and translations for any number of languages.
  - Customizable widgets using object-oriented design, using model-view-presenter pattern.
  - GCC, EWARM, and MDK-ARM compiler support, PC simulator, GCC, and MSVS compiler support for simulator builds.
  - Support of FreeRTOS™ and any CMSIS-OS -compliant operating system out-of-the-box.
  - TouchGFXDesigner, an intuitive WYSIWYG designer tool, with automatic code generation and an automatic IDE project updater (EWARM, MDK-ARM, MSVS).
- LibJPEG:
  - Open-source standard.
  - C implementation for JPEG image encoding and decoding.
- FreeRTOS™:
  - Open-source standard.
  - CMSIS compatibility layer.
  - Tickless operation in low-power mode.
  - Integration with all STM32Cube middleware modules.
- FatFS (FAT file system):
  - FATFS FAT open-source library.
  - Long file name support.
  - Dynamic multidrive support.
  - RTOS and standalone operation.
  - Examples with microSD™ and USB host mass-storage class.
- LwIP TCP/IP stack:
  - Open-source standard.
  - RTOS and standalone operation.

It is based on a modular architecture that allows it to be ported easily to any hardware by just implementing the low-level routines.

Examples based on the middleware components

Each middleware component comes with one or more examples (also called applications), showing how to use it. Integration examples that use several middleware components are provided as well.

3.3 Level 2

This level is composed of a single layer, which is a global real-time and graphical demonstration based on the middleware service layer, the low-level abstraction layer, and the applications that make basic use of the peripherals for board-based functions.

4 STM32CubeF4 MCU package overview

4.1 Supported STM32F4 devices and hardware

STM32Cube offers a highly portable hardware abstraction layer (HAL) built around a generic and modular architecture, allowing the upper layers, middleware, and application to implement its functions without in-depth knowledge of the MCU being used. This improves library code reusability and ensures easy portability from one device to another.

STM32CubeF4 offers full support for all STM32F4 series devices. The user only has to define the right macro in the `stm32f4xx.h` file.

Table 1 lists which macro to define depending on the STM32F4 device used (this macro can also be defined in the compiler preprocessor).

Table 1. **Macros for STM32F4 series**

Macro defined in stm32f4xx.h	STM32F4 devices
STM32F405xx	STM32F405RG, STM32F405VG, STM32F405ZG
STM32F415xx	STM32F415RG, STM32F415VG, STM32F415ZG
STM32F407xx	STM32F407VG, STM32F407VE, STM32F407ZG, STM32F407ZE, STM32F407IG, STM32F407IE
STM32F417xx	STM32F417VG, STM32F417VE, STM32F417ZG, STM32F417ZE, STM32F417IG, STM32F417IE
STM32F427xx	STM32F427VG, STM32F427VI, STM32F427ZG, STM32F427ZI, STM32F427IG, STM32F427II
STM32F437xx	STM32F437VG, STM32F437VI, STM32F437ZG, STM32F437ZI, STM32F437IG, and STM32F437II
STM32F429xx	STM32F429VG, STM32F429VI, STM32F429ZG, STM32F429ZI, STM32F429BG, STM32F429BI, STM32F429NG, STM32F439NI, STM32F429IG, STM32F429II
STM32F439xx	STM32F439VG, STM32F439VI, STM32F439ZG, STM32F439ZI, STM32F439BG, STM32F439BI, STM32F439NG, STM32F439NI, STM32F439IG, STM32F439II
STM32F401xC	STM32F401CB, STM32F401CC, STM32F401RB, STM32F401RC, STM32F401VB, STM32F401VC
STM32F401xE	STM32F401CD, STM32F401RD, STM32F401VD, STM32F401CE, STM32F401RE, STM32F401VE
STM32F411xE	STM32F411CC, STM32F411RC, STM32F411VC, STM32F411CE, STM32F411RE, STM32F411VE
STM32F446xx	STM32F446MC, STM32F446ME, STM32F446RC, STM32F446RE, STM32F446VC, STM32F446VE, STM32F446ZC, STM32F446ZE
STM32F469xx	STM32F469AI, STM32F469II, STM32F469BI, STM32F469NI, STM32F469AG, STM32F469IG, STM32F469BG, STM32F469NG, STM32F469AE, STM32F469IE, STM32F469BE, STM32F469NE
STM32F479xx	STM32F479AI, STM32F479II, STM32F479BI, STM32F479NI, STM32F479AG, STM32F479IG, STM32F479BG, STM32F479NG
STM32F410Rx	STM32F410R8, STM32F410RB
STM32F410Cx	STM32F410C8, STM32F410CB
STM32F410Tx	STM32F410T8, STM32F410TB
STM32F412Cx	STM32F412CG, STM32F412CE
STM32F412Rx	STM32F412RG, STM32F412RE
STM32F412Vx	STM32F412VG, STM32F412VE
STM32F412Zx	STM32F412ZG, STM32F412ZE
STM32F413xx	STM32F413xG, STM32F413xH

Macro defined in stm32f4xx.h	STM32F4 devices
STM32F423xx	STM32F423CH, STM32F423RH, STM32F423VH, STM32F423ZH

STM32CubeF4 features a rich set of examples and demonstrations at all levels, making it easy to understand and use HAL drivers and/or middleware components. These examples run on the STMicroelectronics boards listed in the following table:

Table 2. **Evaluation and Discovery boards for STM32F4 series**

Board	STM32F4 devices supported
STM32429I-EVAL	STM32F429xx
STM32439I-EVAL	STM32F439xx
STM3240G-EVAL	STM32F407xx
STM3241G-EVAL	STM32F417xx
STM32F4DISCOVERY	STM32F407xx
32F401CDISCOVERY	STM32F401xC
32F429IDISCOVERY	STM32F429xx
NUCLEO-F401RE	STM32F401xE
NUCLEO-F411RE	STM32F411xE
STM32446E-EVAL	STM32F446xx
NUCLEO-F446xE	STM32F446xx
STM32469I-EVAL	STM32F469xx
32F469IDISCOVERY	STM32F469xx
NUCLEO-F410RB	STM32F410xx
NUCLEO-F429ZI	STM32F429ZI
NUCLEO-F446ZE	STM32F446ZE
32F411EDISCOVERY	STM32F411xE
32F412GDISCOVERY	STM32F412Zx
NUCLEO-F412ZG	STM32F412Zx
NUCLEO-F413ZH	STM32F413ZH
32F413HDISCOVERY	STM32F413ZH

STM32F4 supports both Nucleo-64 and Nucleo-144 boards. These boards support Adafruit LCD, the ARDUINO® Uno shields that embed a microSD™ connector and a joystick in addition to the LCD.

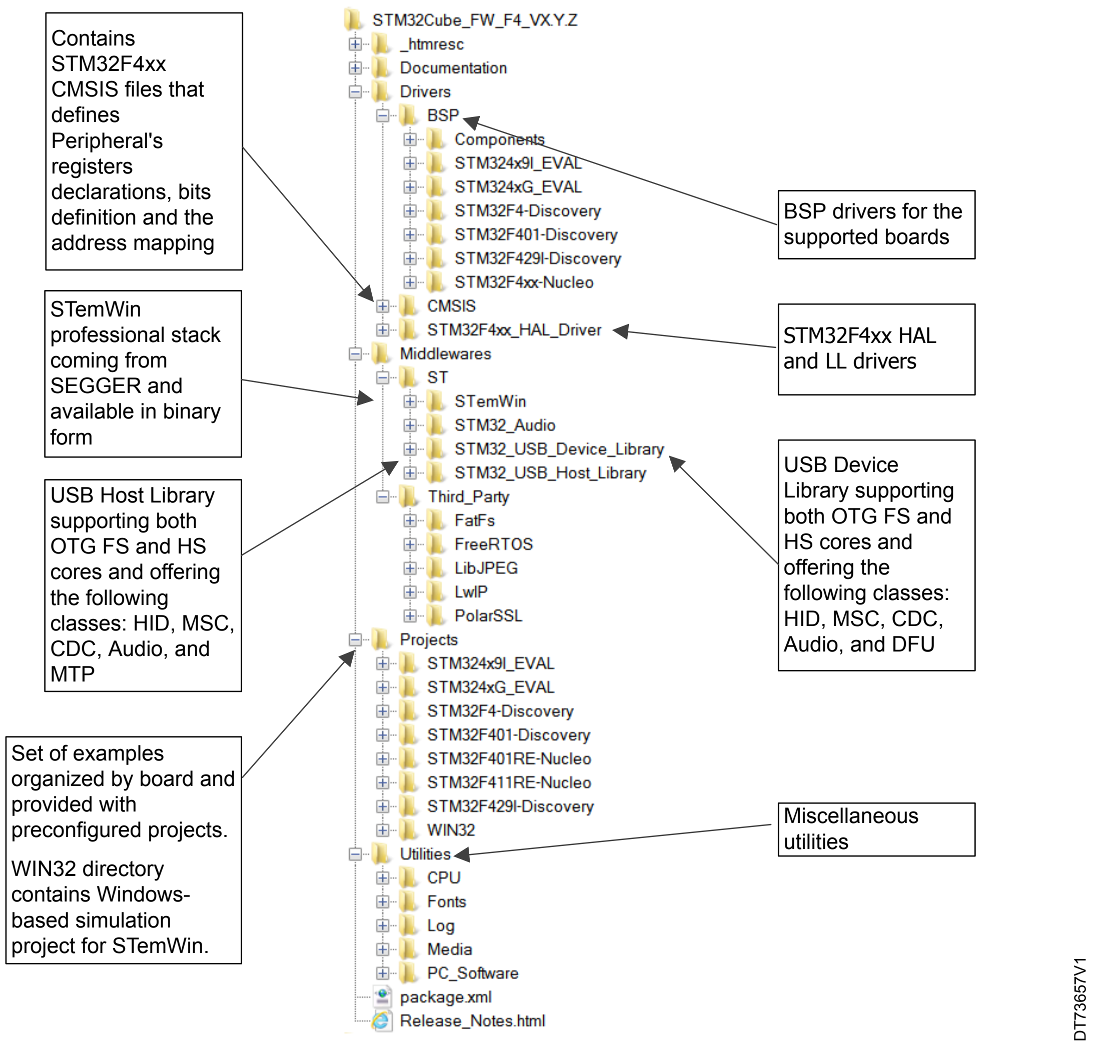
The ARDUINO® shield drivers are provided within the BSP component. Their usage is illustrated by a demonstration firmware.

The STM32CubeF4 firmware can run on any compatible hardware. If the user's board has the same hardware features as the STMicroelectronics board (LED, LCD display, push-buttons, etc.), the user only has to update the BSP drivers to port the provided examples onto their board.

4.2 MCU package overview

The STM32CubeF4 firmware solution is provided in a single zip package with the structure shown in Figure 3 below.

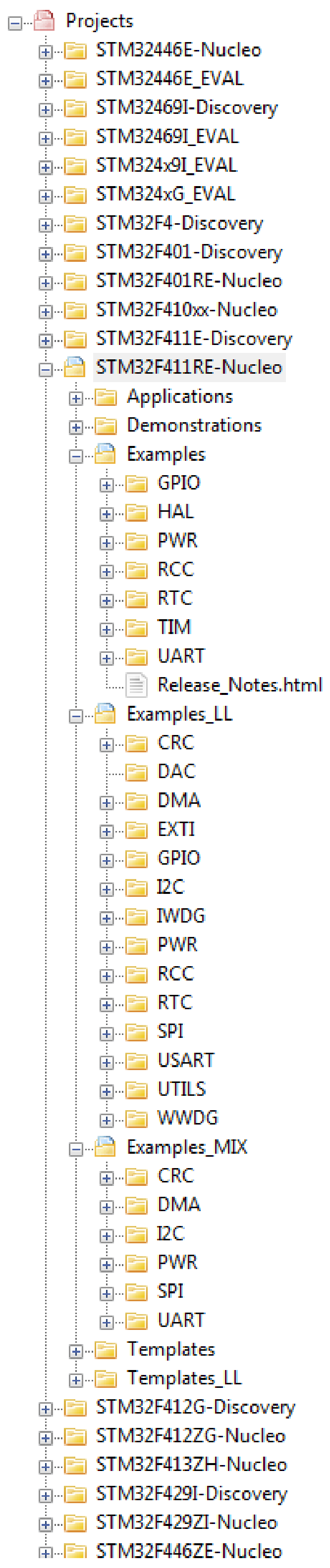
Figure 3. STM32CubeF4 MCU package structure


STM32Cube_FW_F4_VX.Y.Z
_htmresc
Documentation
Drivers
BSP
Components
STM324x9I_EVAL
STM324xG_EVAL
STM32F4-Discovery
STM32F401-Discovery
STM32F429I-Discovery
STM32F4xx-Nucleo
CMSIS
STM32F4xx_HAL_Driver
Middlewares
ST
STemWin
STM32_Audio
STM32_USB_Device_Library
STM32_USB_Host_Library
Third_Party
FatFs
FreeRTOS
LibJPEG
LwIP
PolarSSL
Projects
STM324x9I_EVAL
STM324xG_EVAL
STM32F4-Discovery
STM32F401-Discovery
STM32F401RE-Nucleo
STM32F411RE-Nucleo
STM32F429I-Discovery
WIN32
Utilities
CPU
Fonts
Log
Media
PC_Software
package.xml
Release_Notes.html
Contains STM32F4xx 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
USB Host Library supporting both OTG FS and HS cores and offering the following classes: HID, MSC, CDC, Audio, and MTP
Set of examples organized by board and provided with preconfigured projects.
WIN32 directory contains Windows-based simulation project for STemWin.
BSP drivers for the supported boards
STM32F4xx HAL and LL drivers
USB Device Library supporting both OTG FS and HS cores and offering the following classes: HID, MSC, CDC, Audio, and DFU
Miscellaneous utilities
DT73657V1

For each board, a set of examples is provided with preconfigured projects for EWARM, MDK-ARM™, and STM32CubeIDE toolchains.

Figure 4 shows the project structure for the STM32F411RE-Nucleo board. The structure is identical for other boards.

Figure 4. **STM32CubeF4 examples overview**



```
Projects
  STM32446E-Nucleo
  STM32446E_EVAL
  STM32469I-Discovery
  STM32469I_EVAL
  STM324x9I_EVAL
  STM324xG_EVAL
  STM32F4-Discovery
  STM32F401-Discovery
  STM32F401RE-Nucleo
  STM32F410xx-Nucleo
  STM32F411E-Discovery
  STM32F411RE-Nucleo
    Applications
    Demonstrations
    Examples
      GPIO
      HAL
      PWR
      RCC
      RTC
      TIM
      UART
      Release_Notes.html
    Examples_LL
      CRC
      DAC
      DMA
      EXTI
      GPIO
      I2C
      IWDG
      PWR
      RCC
      RTC
      SPI
      USART
      UTILS
      WWDG
    Examples_MIX
      CRC
      DMA
      I2C
      PWR
      SPI
      UART
    Templates
    Templates_LL
  STM32F412G-Discovery
  STM32F412ZG-Nucleo
  STM32F413ZH-Nucleo
  STM32F429I-Discovery
  STM32F429ZI-Nucleo
  STM32F446ZE-Nucleo
```

The examples are classified depending on the STM32Cube level they apply to, and are named as follows:

- Examples in level 0 are:
  - Examples using HAL drivers.
  - Examples_LL, using LL drivers.
  - Examples_MIX, using a mix of HAL and LL drivers, without any middleware component.
- Examples in level 1 are called Applications and provide typical use cases of each middleware component.
- Examples in level 2 are called Demonstrations and implement the HAL, BSP, and middleware components.

A template project is provided to allow the user to quickly build any firmware application on a given board.

All examples have the same structure:

- An `\\Inc` folder that contains all header files.
- A `\\Src` folder for the sources code.
- `\\EWARM`, `\\MDK-ARM`, and `\\STM32CubeIDE` folders that contain the preconfigured project for each toolchain.
- A `readme.txt` file that describes the example behavior and the environment required to make it work.

Table 3 below provides the number of examples, applications, and demonstrations available for each board.

Table 3. Number of examples and applications available for each board

Board	Templates	Templates_LL	Examples	Examples_LL	Examples_MIX	Applications	Demonstration
STM32429IEVAL STM32439IEVAL	1	1	87	NA	NA	79	3
STM3240GEVAL STM3241GEVAL	1	1	72	NA	NA	54	1
STM32F4DISCOVERY	1	1	27	NA	NA	3	1
32F401CDISCOVERY	1	1	24	NA	NA	3	1
NUCLEO-F401RE	1	1	7	NA	NA	NA	NA
NUCLEO-F411RE	1	1	9	62	12	NA	1
32F411EDISCOVERY	1	1	24	NA	NA	3	1
32F429IDISCOVERY	1	1	30	NA	NA	28	3
NUCLEO-F429ZI	1	1	27	7	1	7	1
STM32446E-EVAL	1	1	71	NA	NA	38	1
NUCLEO-F446ZE	1	1	27	NA	NA	6	1
STM32469I-EVAL	1	1	94	NA	NA	64	3
32F469IDISCOVERY	1	1	46	NA	NA	46	3
NUCLEO-F410RB	1	1	16	2	NA	1	NA
32F412GDISCOVERY	1	1	50	NA	NA	24	1
NUCLEO-F412ZG	1	1	42	NA	NA	10	1
NUCLEO-F413ZH	1	1	48	NA	NA	18	1
32F413HDISCOVERY	1	1	27	NA	NA	20	1

5 Getting started with STM32CubeF4

5.1 How to run a first example

This section explains the simple steps to run a first example with STM32CubeF4. It describes how to run a simple LED-toggling example on the STM32F4-Discovery board:

1. After downloading the STM32CubeF4 MCU package, unzip it into a selected directory. Ensure that the package structure was not modified (as shown in Figure 3 above).
2. Browse to `\\Projects\\STM32F4-Discovery\\Examples`.
3. Open `\\GPIO`, then the `\\GPIO_EXTI` folder.
4. Open the project with the preferred toolchain.
5. Rebuild all files and load the image into the target memory.
6. Run the example. Each time the user button 4 is pressed, the LED toggles (for more details, refer to the example *readme* file).

Below is a quick overview of how to open, build, and run an example with the supported toolchains.

- <u>EWARM:</u>
  1. In the example folder, open the `\\EWARM` subfolder.
  2. Open the `Project.eww` workspace (the workspace name can be different from one example to another).
  3. Rebuild all files: [**Project**]>[**Rebuild all**].
  4. Load the project image: [**Project**]>[**Debug**].
  5. Run program: [**Debug**]>[**Go**] (F5).
- <u>MDK-ARM:</u>
  1. In the example folder, open the `\\MDK-ARM` subfolder.
  2. Open the `Project.uvproj` workspace (the workspace name can be different from one example to another).
  3. Rebuild all files: [**Project**]>[**Rebuild all target files**].
  4. Load the project image: [**Debug**]>[**Start/Stop Debug Session**].
  5. Run program: [**Debug**]>[**Run**] (F5).
- <u>STM32CubeIDE:</u>
  1. Open the STM32CubeIDE toolchain.
  2. Click on [**File**]>[**Switch Workspace**]>[**Other**] and browse to the STM32CubeIDE workspace directory.
  3. Click on [**File**]>[**Import**], select [**General**]>[**Existing Projects into Workspace**] and click [**Next**].
  4. Browse to the STM32CubeIDE workspace directory and select the project.
  5. Rebuild all project files: select the project in the Project explorer window and click on [**Project**]>[**Build project**].
  6. Run program: [**Run**]>[**Debug**] (F11).

5.2 How to develop an application

This section describes the steps required to create an application using STM32CubeF4.

5.2.1 HAL application

1. **Create a project.**
   To create a new project, start with the template project provided for each board in the `\\Projects\\<STM32xx_xxx>\\Templates` folder or from any available project in the `\\Projects\\<STM32xx_xxx>\\Examples` or `\\Projects\\<STM32xx_xxx>\\Applications` folder (*<STM32xx_xxx>* refers to the board name, for example, STM32F4-Discovery).
   The template project provides an empty main loop function, which allows the user to get familiar with the project settings for STM32CubeF4:
   a. It contains sources of the HAL, CMSIS, and BSP drivers, which are the minimum required components to develop code for any given board.
   b. It contains the include paths for all firmware components.
   c. It defines the STM32F4 device supported, ensuring the correct configuration for the CMSIS and HAL drivers.
   d. It provides ready-to-use user files, preconfigured as follows:
      - HAL is initialized.
      - SysTick ISR implemented for *HAL_Delay()* purposes.
      - The system clock is configured with the maximum frequency of the device.

Note: *If an existing project is copied into another location, the include path must be updated.*

2. **Add the necessary middleware to the project (optional).**
   The available middleware stacks are USB host and device libraries, STemWin, ST-TouchGFX, LibJPEG, FreeRTOS™, FatFS, LwIP, and mbedTLS. To find out which source files must be added to the project file list, refer to the documentation provided for each middleware stack. The user can also look at the applications available in the `\\Projects\\STM32xx_xxx\\Applications\\<MW_Stack>` folder (*<MW_Stack>* refers to the middleware stack, for example, USB_Device) to get a better idea of the source files to be added and the include paths.
3. **Configure the firmware components.**
   The HAL and middleware components offer a set of build-time configuration options using macros declared with *#define* in a header file. A template configuration file is provided within each component, which the user has to copy to the project folder (usually the configuration file is named `xxx_conf_template.h`. The word *_template* needs to be removed when copying it to the project folder). The configuration file provides enough information to know the effect of each configuration option. More detailed information is available in the documentation provided for each component.
4. **Start the HAL library.**
   After jumping to the main program, the application code needs to call the *HAL_Init()* API to initialize the HAL library, which does the following:
   a. Configure the flash memory prefetch, and instruction and data caches (user-configurable by macros defined in `stm32f4xx_hal_conf.h`).
   b. Configures SysTick to generate an interrupt every 1 ms. SysTick is clocked by the HIS oscillator (default configuration after reset).
   c. Set NVIC group priority to 4.
   d. Call the *HAL_MspInit()* callback function defined in the user file `stm32f4xx_hal_msp.c` to initialize the global low-level hardware.
5. **Configure the system clock.**
   The system clock configuration is done by calling these two APIs:
   – *HAL_RCC_OscConfig()*: configures the internal and/or external oscillators, PLL source, and factors. The user can choose to configure one or all oscillators. If the system must not run at a high frequency, the user can skip the PLL configuration.
   – *HAL_RCC_ClockConfig()*: configures the system clock source, flash memory latency, and AHB and APB prescalers.

6. **Peripheral initialization**
   a. Start by writing the peripheral *HAL_PPP_MspInit* function. For this function, proceed as follows:
      ◦ Enable the peripheral clock.
      ◦ Configure the peripheral GPIOs.
      ◦ Configure the DMA channel and enable DMA interrupt (if necessary).
      ◦ Enable peripheral interrupt (if necessary).
   b. Edit the `stm32f4xx_it.c` file to call the required interrupt handlers (peripheral and DMA), if necessary.
   c. Write process complete callback functions if the user plans to use peripheral interrupt or DMA.
   d. In the `main.c` file, initialize the peripheral handle structure and call the *HAL_PPP_Init()* function to initialize the peripheral.
7. **Develop an application process.**
   At this stage, the system is ready and the user can start developing the application code.
   a. The HAL provides intuitive and ready-to-use APIs for configuring the peripheral, and supports polling, interrupt, and DMA programming models to accommodate any application requirements. For more details on how to use each peripheral, refer to the rich example set provided.
   b. If the application has any real-time constraints, the user can find a large set of examples showing how to use the FreeRTOS™ and integrate it with the middleware stacks provided in STM32CubeF4. This can be a good starting point for a first development.

Note: In the default HAL implementation, the SysTick timer is the timebase source. It is used to generate interrupts at regular time intervals. If HAL_Delay() is called from the 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 timebase configurations are declared as __Weak to make overrides possible in case of other implementations in the user file (using a general purpose timer, for example, or another time source). For more details, refer to the HAL_TimeBase example.

5.2.2 LL application

This section describes the steps required to create your own LL application using STM32CubeF4.

- **Create your project.**
  To create a new project, either start with the *Templates_LL* project provided for each board in the `\\Projects\\<STM32xxx_yyy>\\Templates_LL` folder, or with any available project in the `\\Projects\\<STM32xxy_yyy>\\Examples_LL` folder (*<STM32xxx_yyy>* refers to the board name, such as STM32F411RE-Nucleo).
  The template project provides an empty main loop function, making it a good starting point to get familiar with the project settings for STM32CubeF4.
  The main template characteristics are as follows:
  – It contains the source code of LL and CMSIS drivers, which are the minimal components to develop code on any given board.
  – It contains the include paths for all the required firmware components.
  – It selects the supported STM32F4 device and allows the configuration of the CMSIS and LL drivers accordingly.
  – It provides ready-to-use user files that are preconfigured as follows:
    ◦ `main.h`: LED & USER_BUTTON definition abstraction layer.
    ◦ `main.c`: system clock configuration for maximum frequency.

- **Port an existing project to another board.**
  To port an existing project to another target board, start with the *Templates_LL* project provided for each board and available in the `\\Projects\\<STM32xxx_yyy>\\Templates_LL` folder.
  1. Select an LL example. To find the board on which LL examples are deployed, refer to the list of LL examples in `STM32CubeProjectsList.html` and to Table 3. Number of examples and applications available for each board.
  2. Port the LL example:
     a. Copy/paste the `Templates_LL` folder to keep the initial source or directly update the existing *Templates_LL* project.
     b. Replace the *Templates_LL* files with the *Examples_LL* targeted project files.
     c. Keep all board specific parts. For reasons of clarity, board specific parts have been flagged with specific tags:

     ```
     /* =========== BOARD SPECIFIC CONFIGURATION CODE BEGIN =========== */
     /* ============ BOARD SPECIFIC CONFIGURATION CODE END ============= */
     ```

     Thus the main porting steps are the following:
     i. Replace the `stm32f4xx_it.h` file.
     ii. Replace the `stm32f4xx_it.c` file.
     iii. Replace the `main.h` file and update it. Keep the LED and user button definition of the LL template under *"BOARD SPECIFIC CONFIGURATION"* tags.
     iv. Replace the `main.c` file, and update it. Keep the clock configuration of the *SystemClock_Config()* function under *"BOARD SPECIFIC CONFIGURATION"* tags.
     v. Depending on the LED definition, replace all LEDx occurrences with another LEDy available in `main.h`.

  With these adaptations, the example should be functional on the targeted board.

5.3 Using STM32CubeMX for generating the initialization C code

An alternative to steps 1 to 6 described in the Section 5.2.1 consists of using the STM32CubeMX tool to generate code for the initialization of the system, peripherals, and middleware through a step-by-step process:

1. Selection of the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
2. Configuration of each required embedded software tool using a pinout-conflict solver, clock-tree setting helper, power consumption calculator, and utility performing MCU peripheral configuration (GPIO, USART, ...) and middleware stacks (USB, TCP/IP, ...).
3. Generation of the initialization C code based on the configuration selected. This code is ready to be used within several development environments. The user code is kept at the next code generation.

For more information, refer to the *STM32CubeMX for STM32 configuration and initialization C code generation* user manual (UM1718).

5.4 How to get STM32CubeF4 release updates

The STM32CubeF4 MCU package comes with an updater utility: STM32CubeUpdater, also available as a menu within the STM32CubeMX code generation tool.

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

5.4.1 How to install and run the STM32CubeUpdater program

1. Double-click the `SetupSTM32CubeUpdater.exe` file to launch the installation.
2. 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.
3. Right-click the updater icon and select [**Updater Settings**] to configure the updater connection and whether to perform manual or automatic checks (refer to Section 3 of *STM32CubeMX user guide* (UM1718) for more details on updater configuration).

6 FAQs

What is the license scheme for the STM32CubeF4 firmware?

The HAL is distributed under a nonrestrictive BSD (Berkeley Software Distribution) license.

The middleware stacks made by STMicroelectronics (USB host and device libraries, STemWin, ST-TouchGFX) come with a licensing model allowing easy reuse, provided it runs on an STMicroelectronics device.

The middleware, based on well-known open-source solutions (FreeRTOS™, FatFS, LwIP, and mbedTLS), has user-friendly license terms. For more details, refer to the license agreement of each middleware.

Which boards are supported by the STM32CubeF4 MCU Package?

The STM32CubeF4 MCU Package provides BSP drivers and ready-to-use examples for the following STM32F4 boards: STM324x9I-EVAL, STM324xG-EVAL, STM32446E-EVAL, STM32F4-Discovery, STM32F401-Discovery, STM32F429I-Discovery, STM32F4xx-Nucleo, STM32469I-EVAL STM32469I-Discovery, STM32F446E-Nucleo, STM32F410xx-Nucleo, STM32F429ZI-Nucleo, STM32F446ZE-Nucleo, STM32F411E-Discovery, STM32F412G-Discovery, STM32F412ZG-Nucleo, STM32F413H-Discovery, STM32F413ZH-Nucleo.

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

Yes, it does. 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, there are. STM32CubeF4 provides a rich set of examples and applications (140 for STM324x9I-EVAL, for example). 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, that is, 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 built-in knowledge of STM32 microcontrollers, including their peripherals and software. This allows 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 STM32CubeF4 firmware releases?

The STM32CubeF4 MCU Package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new MCU Package updates (new releases or/and patches).

STM32CubeUpdater is integrated as well within the STM32CubeMX tool. When using this tool for STM32F4 configuration and initialization C code generation, the user can benefit from the STM32CubeMX self-updates, as well as STM32CubeF4 MCU Package updates.

For more details, refer to Section 5.4 How to get STM32CubeF4 release updates.

Is there any link with Standard Peripheral Libraries?

The STM32Cube HAL and LL drivers replace the standard peripheral libraries:

- The HAL drivers offer a higher abstraction level compared to the standard peripheral APIs. They focus on common peripheral features rather than hardware. Their higher abstraction level allows the definition of a set of user-friendly APIs that can be easily ported from one product to another.
- The LL drivers offer low-level APIs at register level. They are organized in a simpler and clearer way than direct register access. LL drivers also include peripheral initialization APIs, which are more optimized compared to what is offered by the SPL, while being functionally similar. Compared to HAL drivers, these LL initialization APIs allow an easier migration from the SPL to the STM32Cube LL drivers, since each SPL API has its equivalent LL API.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IP complexity is hidden for end users.

LL drivers offer low-level APIs at register level, with a better optimization but less portability. They require a deep knowledge of product/IP specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL drivers?

There is no configuration file. The source code must directly include the necessary `stm32f4xx_ll_ppp.h` file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers.

The major difference between HAL and LL is that HAL drivers require the creation and use of handles for operation management, while LL drivers operate directly on peripheral registers. Mixing HAL and LL is illustrated in the Examples_MIX example.

Are there any LL APIs that are not available with HAL?

A few Arm® Cortex® APIs have been added to `stm32f4xx_ll_cortex.h`, for example, for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions require SysTick interrupts to manage timeouts.

How are LL initialization APIs enabled?

The definition of LL initialization APIs and associated resources (structure, literals, and prototypes) is conditioned by the *USE_FULL_LL_DRIVER* compilation switch.

To be able to use LL APIs, add this switch to the toolchain compiler preprocessor.

Revision history

Table 4. **Document revision history**

Date	Revision	Changes
17-Feb-2014	1	Initial release.
25-Jun-2014	2	Added support for STM32F411xD/E part numbers. Updated Section 2 STM32CubeF4 main features. Added LibJPEG in Section 3 STM32CubeF4 architecture overview. Updated Table 2. Evaluation and Discovery boards for STM32F4 series, Table 3. Number of examples and applications available for each board, Figure 3. STM32CubeF4 MCU package structure. Updated Section 5.2 How to develop an application and Section 5.3 Using STM32CubeMX for generating the initialization C code.
10-Mar-2015	3	Updated Figure 1. STM32CubeF4 firmware components. Added support for STM32F446xx devices: • Updated Table 1 and Table 2. Evaluation and Discovery boards for STM32F4 series and Table 3. Number of examples and applications available for each board. • Updated Section 6 FAQs.
19-May-2015	4	Added support for SW4STM32 toolchain.
21-Sep-2015	5	Added support for STM32F469xx STM32F410xx. Updated Table 1 and Table 2. Evaluation and Discovery boards for STM32F4 series.
12-Nov-2015	6	Added support for STM32F411xE. Updated Table 1 and Table 2. Evaluation and Discovery boards for STM32F4 series and Table 3. Number of examples and applications available for each board.
10-May-2016	7	Added support for STM32F412xx. Updated Table 1 and Table 2. Evaluation and Discovery boards for STM32F4 series and Table 3. Number of examples and applications available for each board.
02-Nov-2016	8	Added support for STM32F413xx and STM32F423xx. Updated Table 1 and Table 2. Evaluation and Discovery boards for STM32F4 series and Table 3. Number of examples and applications available for each board.
16-Feb-2017	9	Added support for LL drivers on cover page. Added Section 5.2.1 HAL application and Section 5.2.2 LL application. Updated: • Figure 1. STM32CubeF4 firmware components. • Figure 2. STM32CubeF4 firmware architecture. • Figure 3. STM32CubeF4 MCU package structure. • Section 4.2 MCU package overview. • Table 3. Number of examples and applications available for each board. • Added new FAQs in Section 6 FAQs.
17-Feb-2017	10	Corrected date for revision 9 in document revision history table.
02-Feb-2018	11	Not applicable, superseded by revision 13.
13-Feb-2018	12	Internal version.
22-Mar-2018	13	No major change versus revision 10. Minor modifications: • Board numbers corrected in Table 2. Evaluation and Discovery boards for STM32F4 series and Table 3. Number of examples and applications available for each board. • Changed LL API preferred to ‘low-layer APIs in Figure 1. STM32CubeF4 firmware components. • Document identifier replaced by UM1730 in page footer. • Replaced “firmware package” by “MCU Package” in the whole document.

Date	Revision	Changes
05-Feb-2019	14	Changed STM32Cube logo on cover page. Removed all references to TrueSTUDIO. Added description of ST-TouchGFX graphical stack in Section 2 STM32CubeF4 main features, Section 3 STM32CubeF4 architecture overview; Section 5.2.1 HAL application and Section 6 FAQs.
18-Dec-2023	15	Updated document title. Changed SW4STM32 references to STM32CubeIDE. General document cleanup.

Contents

List of tables

List of figures