UM1730
User manual

Getting started with STM32CubeF4 firmware package for STM32F4 Series

Introduction

The STMCube™ initiative was originated by STMicroelectronics to help reduce development effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for the STM32F4 series)
  - The STM32Cube HAL STM32 abstraction-layer embedded software, ensuring maximized portability across the STM32 portfolio.
  - Low Layer APIs (LL) offering a fast, lightweight, expert-oriented layer which is closer to the hardware than the HAL. The LL APIS are available for a set of peripherals.
  - A consistent set of middleware components, such as RTOS, USB, TCP/IP and graphics.
  - All embedded software utilities, with a full set of examples.

This user manual describes how to get started with the STM32CubeF4 firmware package.

Section 1 describes the main features of STM32CubeF4 firmware, part of the STMCube™ initiative.

Section 2 and *Section 3* provide an overview of the STM32CubeF4 architecture and firmware package structure.

Contents

List of tables

List of figures

1 STM32CubeF4 main features

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

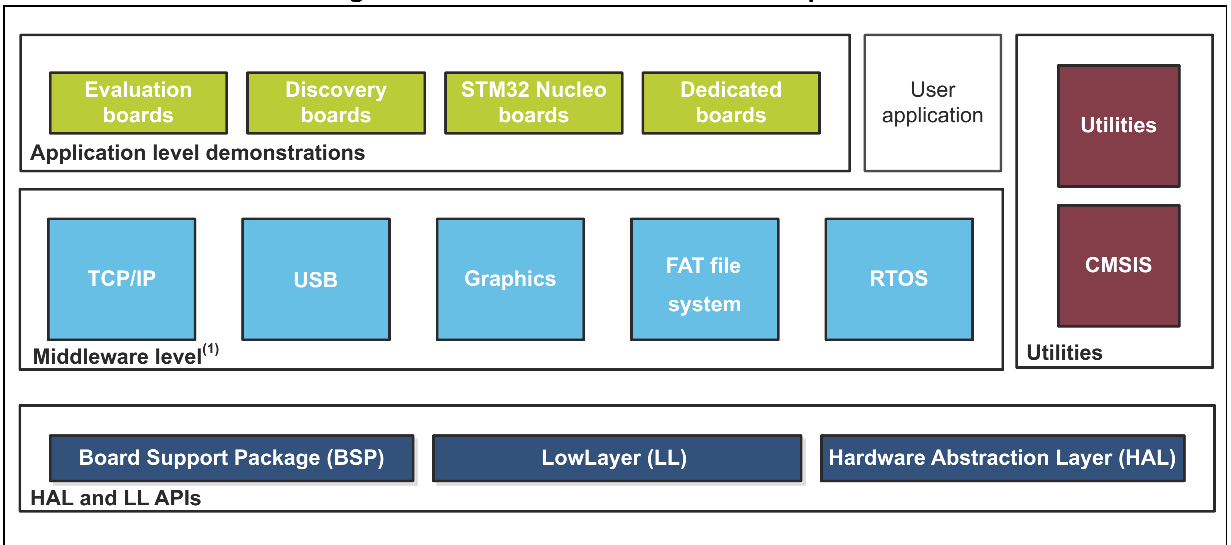
STM32CubeF4 is fully compatible with the STM32CubeMX code generator that allows the user to generate initialization code. The package includes Low Layer (LL) and Hardware Abstraction Layer (HAL) APIs that cover the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL and LL APIs are available in an open-source BSD license for user convenience.

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

- 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 ST partner SEGGER
  - LibJPEG, an open source implementation on STM32 for JPEG images encoding and decoding
- CMSIS-RTOS implementation with FreeRTOS open source solution
- FAT file system based on open source FatFS solution
- TCP/IP stack based on open source LwIP solution
- SSL/TLS secure layer based on open source mbedTLS

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

Figure 1. STM32CubeF4 firmware components

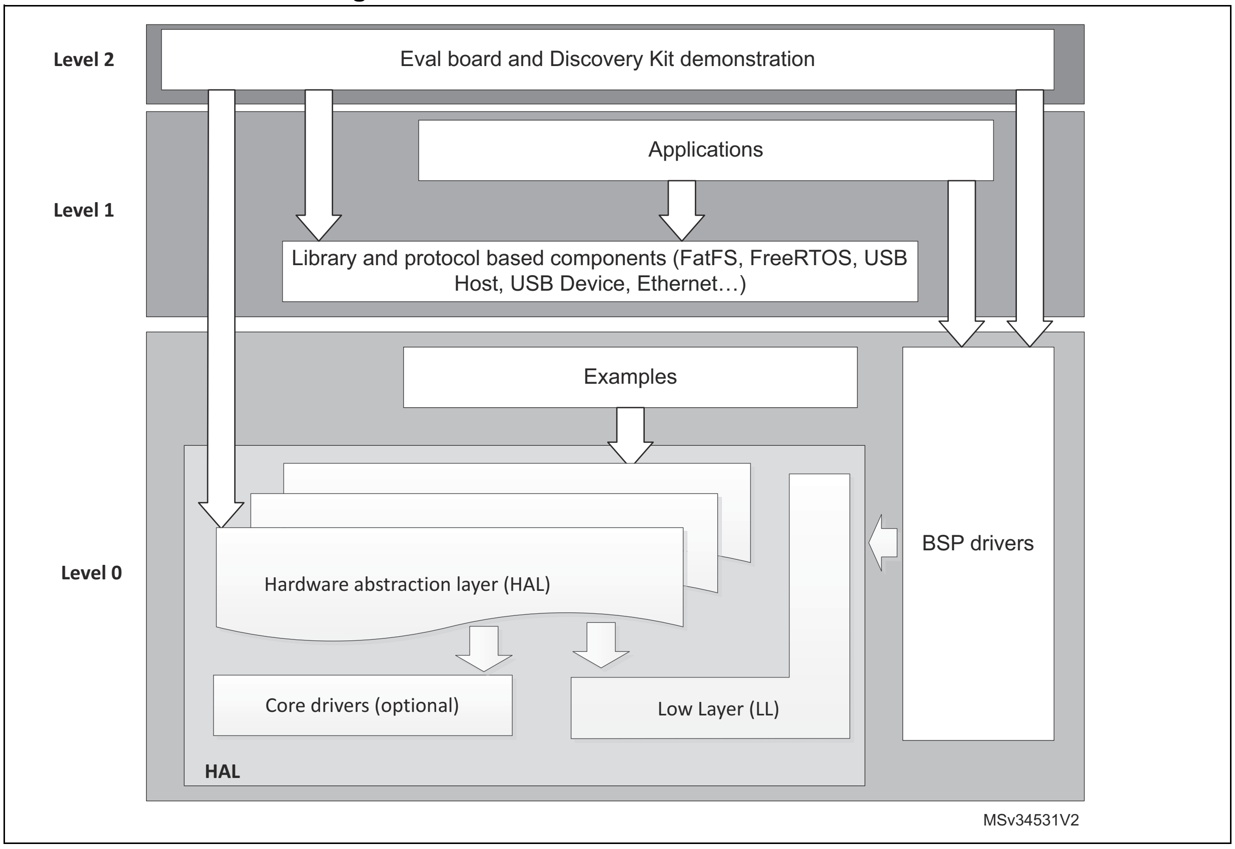


(1) The set of middleware components depends on the product Series.

2 STM32CubeF4 architecture overview

The STM32CubeF4 firmware solution is built around three independent levels that can easily interact with each other as described in the *Figure 2* below:

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)
BSP drivers
HAL
MSv34531V2

Level 0: this level is divided into three sub-layers:

- 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 composed of two parts:
  - Component: is 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: it 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 which allow to offload the user application implementation by providing ready-to-use processes. For example, for the communication peripherals (I^2S, 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 provide common and generic functions to all the STM32 Series and extension APIs which provide specific and customized functions for a specific family or a specific part number.

- **Basic peripheral usage examples:** this layer contains examples of basic operation of the STM32F4 peripherals using only the HAL and/or the Low-Layer driver APIs as well as the BSP resources.
- **Low Layer (LL)**: The low layer APs provide low-level APIs at register level, with better optimization but less portability. They require a deep knowledge of MCU and peripheral specificatins.The LL drivers are designed to offer a fast light-weight expert oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as FSMC, USB or SDMMC).

  The LL drivers feature :
  - A set of functions to initialize peripheral main features 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
  - Function for peripheral de-initialization (peripheral registers restored to their default values)
  - A set of inline functions for direct and atomic register access
  - Full independence from HAL and capability to be used in standalone mode (without HAL drivers)
  - Full coverage of the supported peripheral features.

Level 1: this level is divided into two sub-layers:

- **Middleware components:** a set of libraries covering USB host and device libraries, STemWin, 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, the 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:

  <u>USB host and device libraries</u>
  - 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)
  - Includes 32-bit aligned data structures to handle DMA-based transfer in High-speed modes
  - Supports multi USB OTG core instances from user level through configuration file (allows operation with more than one USB host/device peripheral)
  - 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

  STemWin graphical stack

  - Professional grade solution for GUI development based on the emWin solution by SEGGER
  - Optimized display drivers
  - Software tools for code generation and bitmap editing (STemWin Builder…)

  LibJPEG

  - Open source standard
  - C implementation for JPEG image encoding and decoding.

  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 and USB host mass-storage class

  LwIP TCP/IP stack

  - Open source standard
  - RTOS and standalone operation

- **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 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.

3 STM32CubeF4 firmware package overview

3.1 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 the library code re-usability and guarantees an easy portability from one device to another.

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

Table 1 below 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 and STM32F405ZG
STM32F415xx	STM32F415RG, STM32F415VG and STM32F415ZG
STM32F407xx	STM32F407VG, STM32F407VE, STM32F407ZG, STM32F407ZE, STM32F407IG and STM32F407IE
STM32F417xx	STM32F417VG, STM32F417VE, STM32F417ZG, STM32F417ZE, STM32F417IG and STM32F417IE
STM32F427xx	STM32F427VG, STM32F427VI, STM32F427ZG, STM32F427ZI, STM32F427IG and STM32F427II
STM32F437xx	STM32F437VG, STM32F437VI, STM32F437ZG, STM32F437ZI, STM32F437IG and STM32F437II
STM32F429xx	STM32F429VG, STM32F429VI, STM32F429ZG, STM32F429ZI, STM32F429BG, STM32F429BI, STM32F429NG, STM32F439NI, STM32F429IG and STM32F429II
STM32F439xx	STM32F439VG, STM32F439VI, STM32F439ZG, STM32F439ZI, STM32F439BG, STM32F439BI, STM32F439NG, STM32F439NI, STM32F439IG and STM32F439II
STM32F401xC	STM32F401CB, STM32F401CC, STM32F401RB, STM32F401RC, STM32F401VB and STM32F401VC
STM32F401xE	STM32F401CD, STM32F401RD, STM32F401VD, STM32F401CE, STM32F401RE and STM32F401VE
STM32F411xE	STM32F411CC, STM32F411RC, STM32F411VC, STM32F411CE, STM32F411RE and STM32F411VE
STM32F446xx	STM32F446MC, STM32F446ME, STM32F446RC, STM32F446RE, STM32F446VC, STM32F446VE, STM32F446ZC, STM32F446ZE
STM32F469xx	STM32F469AI, STM32F469II, STM32F469BI, STM32F469NI, STM32F469AG, STM32F469IG, STM32F469BG, STM32F469NG, STM32F469AE, STM32F469IE, STM32F469BE, STM32F469NE

Table 1. Macros for STM32F4 series (continued)

Macro defined in stm32f4xx.h	STM32F4 devices
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
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 *Table 2* below:

Table 2. Evaluation and Discovery boards for STM32F4 series

Board	STM32F4 devices supported
STM324x9I-EVAL	STM32F429xx and STM32F439xx
STM324xG-EVAL	STM32F407xx and STM32F417xx
STM32F4-Discovery	STM32F407xx
STM32F401-Discovery	STM32F401xC
STM32F429I-Discovery	STM32F429xx
STM32F401RE-Nucleo	STM32F401xE
STM32F411RE-Nucleo	STM32F411xE
STM32446E-EVAL	STM32F446xx
STM32F446E-Nucleo	STM32F446xx
STM32469I-EVAL	STM32F469xx
STM32F469I-Discovery	STM32F469xx
STM32F410x-Nucleo	STM32F410xx
STM32F429ZI-Nucleo	STM32F429ZI
STM32F446ZE-Nucleo	STM32F446ZE
STM32F411E-Discovery	STM32F411xE
STM32F412G-Discovery	STM32F412Zx

Table 2. Evaluation and Discovery boards for STM32F4 series (continued)

Board	STM32F4 devices supported
STM32F412ZG-Nucleo	STM32F412Zx
STM32F413ZH-Nucleo	STM32F413ZH
STM32F413H-Discovery	STM32F413ZH

STM32F4 supports both Nucleo-64 and Nucleo-144 boards. These boards support Adafruit LCD display, the Arduino™ UNO shields which embed a microSD connector and a joystick in addition to the LCD.

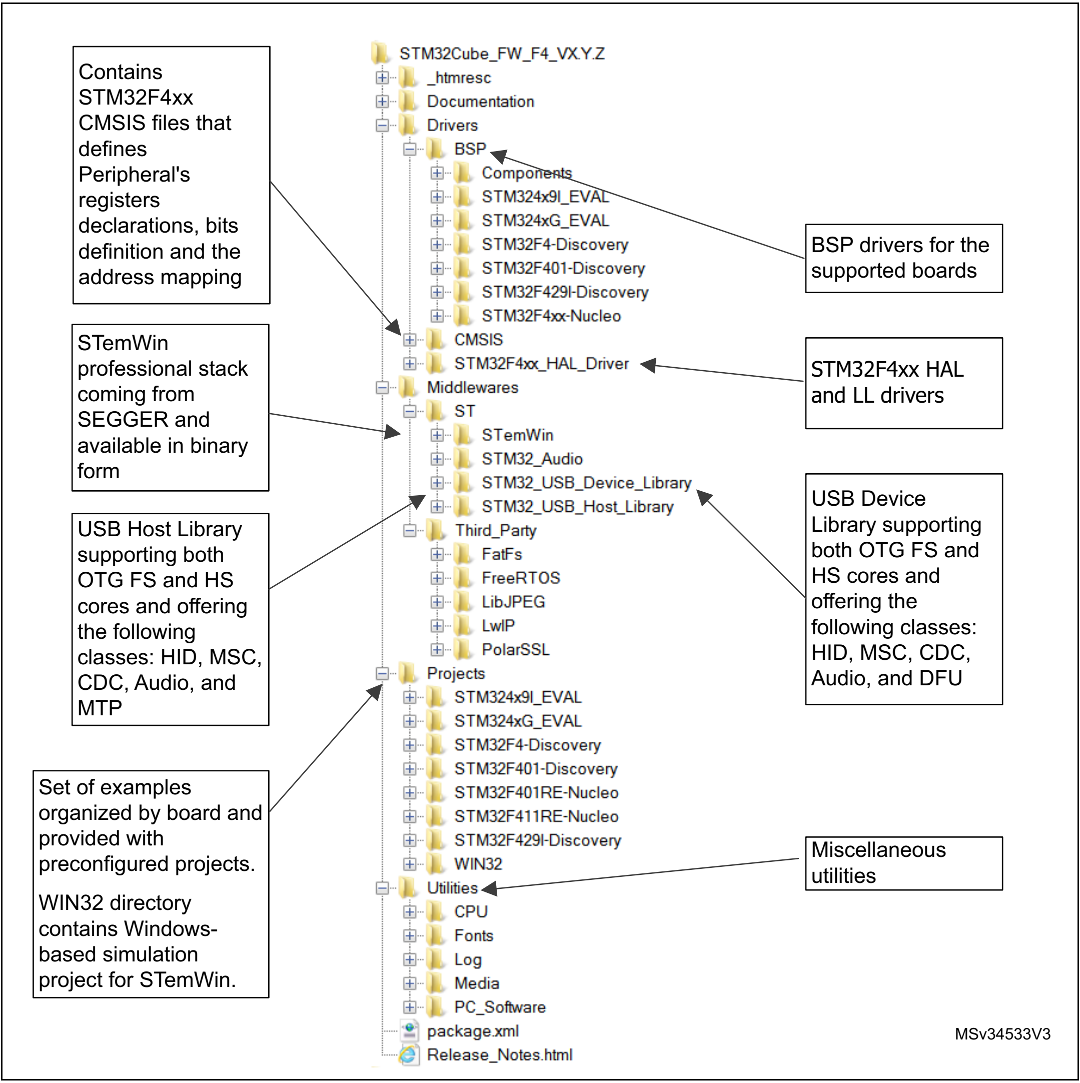
The Arduino™ shield drivers are provided within the BSP component. Their usage is illustrated by a demonstration firmware.

The STM32CubeF4 firmware is able to run on any compatible hardware. If the user's board has the same hardware features as the ST board (LED, LCD display, push-buttons and others), the user has just to update the BSP drivers to port the provided examples on his board.

3.2 Firmware package overview

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

Figure 3. STM32CubeF4 firmware package structure


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.
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
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
MSv34533V3

For each board, a set of examples are provided with preconfigured projects for EWARM, MDK-ARM™, TrueSTUDIO® and SW4STM32 toolchains.

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

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

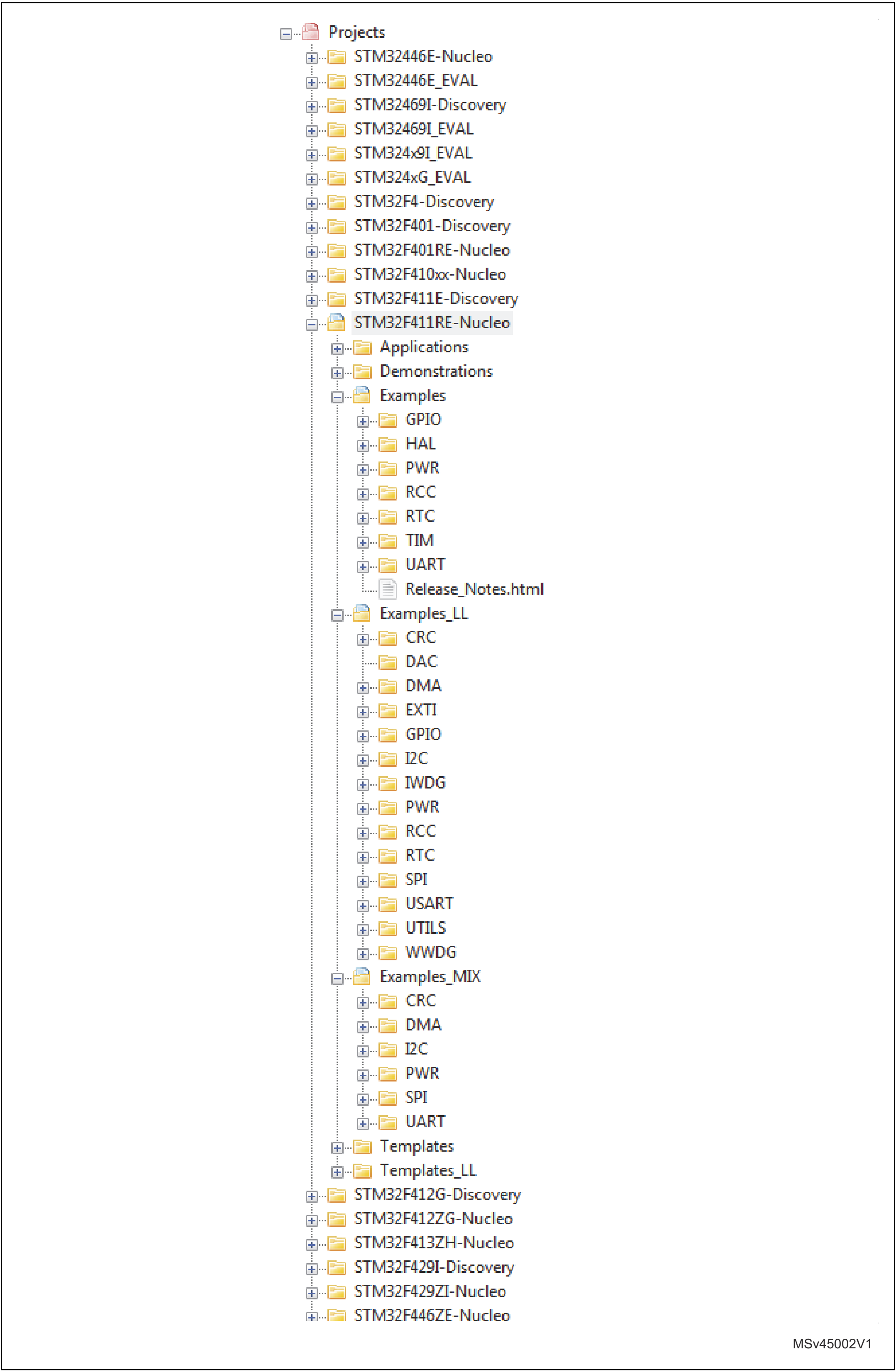
- Examples in level 0 are:
  - Examples using HAL drivers
  - Examples_LL using LL drivers
  - Examples_MIX, using a mix of HAL and LL drivers, without any middleware component
- Examples in level 1 are called Applications and provide typical use cases of each middleware component
- Examples in level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

A template project is provided to allow users 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, \TrueSTUDIO and \SW4STM32 folders contain the preconfigured project for each toolchain
- readme.txt describes the example behavior and the environment required to make it work

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

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

Board	Examples	Examples_LL	Examples_MIX	Applications	Demonstration
STM324x9I-EVAL	87	NA	NA	55	1
STM324xG-EVAL	72	NA	NA	53	1
STM32F4-Discovery	27	NA	NA	3	1
STM32F401-Discovery	24	NA	NA	3	1
STM32F401RE-Nucleo	7	NA	NA	NA	NA
STM32F411RE-Nucleo	9	62	12	NA	1
STM32F411E-Discovery	24	NA	NA	3	1
STM32F429I-Discovery	30	NA	NA	11	1
STM32F429ZI-Nucleo	27	7	1	7	1
STM32446E-EVAL	71	NA	NA	38	1
STM32F446ZE-Nucleo	27	NA	NA	6	1
STM32469I-EVAL	94	NA	NA	43	1
STM32469I-Discovery	46	NA	NA	25	1
STM32F410xx-Nucleo	16	2	NA	1	NA
STM32F412G-Discovery	50	NA	NA	24	1
STM32F412ZG-Nucleo	42	NA	NA	10	1
STM32F413ZH-Nucleo	48	NA	NA	18	1
STM32F413H-Discovery	27	NA	NA	20	1

4 Getting started with STM32CubeF4

4.1 How to run a first example

This section explains how simple it is to run a first example with the STM32CubeF4. As an illustration consider to run a simple LED toggling example running on the STM32F4-Discovery board:

1. After downloading the STM32CubeF4 firmware package, unzip it into a selected directory. Ensure that the package structure is not modified (as shown in the *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).

   User will get a quick overview of how to open, build and run an example with the supported toolchains.

- EWARM
  a) Under the example folder, open the \EWARM subfolder.
  b) Open the Project.eww workspace[a].
  c) Rebuild all files: Project->Rebuild all.
  d) Load project image: Project->Debug.
  e) Run program: Debug->Go (F5).
- MDK-ARM
  a) Under the example folder, open the \MDK-ARM subfolder.
  b) Open the Project.uvproj workspace[a].
  c) Rebuild all files: Project->Rebuild all target files.
  d) Load project image: Debug->Start/Stop Debug Session.
  e) Run program: Debug->Run (F5).
- TrueSTUDIO
  a) Open the TrueSTUDIO toolchain.
  b) Click on File->Switch Workspace->Other and browse to the TrueSTUDIO workspace directory.
  c) Click on File->Import, select General->'Existing Projects into Workspace' and then click “Next”.
  d) Browse to the TrueSTUDIO workspace directory, select the project.
  e) Rebuild all project files: Select the project in the “Project explorer” window then click on Project->build project menu.
  f) Run program: Run->Debug (F11).
- SW4STM32

a. The workspace name may change from one example to another.

a) Open the SW4STM32 toolchain.
b) Click on File->Switch Workspace->Other and browse to the SW4STM32 workspace directory.
c) Click on File->Import, select General->'Existing Projects into Workspace' and then click "Next".
d) Browse to the SW4STM32 workspace directory, select the project.
e) Rebuild all project files: Select the project in the "Project explorer" window then click on Project->build project menu.
f) Run program: Run->Debug (F11).

4.2 How to develop an application

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

4.2.1 HAL application

1. **Create a project:** to create a new project start from the template project provided for each board under \Projects\<STM32xx_xxx>\Templates or from any available project under \Projects\<STM32xx_xxx>\Examples or \Projects\<STM32xx_xxx>\Applications (<STM32xx_xxx> refers to the board name, ex. STM32F4-Discovery).

   The template project provides an empty main loop function, this is a good starting point to allow user to get familiar with the project settings for the STM32CubeF4. It has the following characteristics:

   a) It contains sources of the HAL, CMSIS and BSP drivers which are the minimum required components to develop code for a given board
   b) It contains the include paths for all the firmware components
   c) It defines the STM32F4 device supported, allowing to have the right 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() purpose
      - 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):** available middleware stacks are: USB host and device libraries, STemWin, 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. The user can also have a look at the applications available under \Projects\STM32xx_xxx\Applications\<MW_Stack> (<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) Configures the Flash prefetch, instruction and data caches (user-configurable by macros defined in stm32f4xx_hal_conf.h)
   b) Configure the SysTick to generate an interrupt every 1ms. The SysTick is clocked by the HSI (default configuration after reset)
   c) Sets NVIC Group Priority to 4
   d) Calls the HAL_MspInit() callback function defined in user file stm32f4xx_hal_msp.c to do the global low-level hardware initialization
5. **Configure the system clock:** the system clock configuration is done by calling these two APIs:
   a) HAL_RCC_OscConfig(): configures the internal and/or external oscillators, PLL source and factors. The user can choose to configure one oscillator or all oscillators. If the system must not run at high frequency, the user can skip the PLL configuration
   b) HAL_RCC_ClockConfig(): configures the system clock source, Flash 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 DMA channel and enable DMA interrupt (if needed).
   – Enable peripheral interrupt (if needed).
   b) Edit the stm32f4xx_it.c to call the required interrupt handlers (peripheral and DMA), if needed.
   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, then call the function HAL_PPP_Init() 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 some 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 the STM32CubeF4. It 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 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 override possible in case of other implementations in user file (using a general purpose timer for example or other time source). For more details refer to HAL_TimeBase example.

4.2.2 LL application

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

1. **Create your project**

   To create a new project, you can either start from the *Templates_LL* project provided for each board under \Projects\<STM32xxx_yyy>\Templates_LL or from any available project under \Projects\<STM32xxy_yyy>\Examples_LL (<STM32xxx_yyy> refers to the board name, such as STM32F411RE-Nucleo).
   The *Template* project provides an empty main loop function, however it is a good starting point to get familiar with project settings for STM32CubeF4.

   The main template characteristics are as follows:

   – It contains the source code of LL and CMSIS drivers, that are the minimal components to develop a code on a given board.

   – It contains the include paths for all the required firmware components.

   – It selects the supported STM32F4 device and allows configuring the CMSIS and LL drivers accordingly.

   – It provides ready-to-use user files, that are pre-configured as follows:
   main.h: LED & USER_BUTTON definition abstraction layer
   main.c: System clock configuration for maximum frequency.

2. **Port an existing project to another board**

   To port an existing project to another target board, start from the *Templates_LL* project provided for each board and available under \Projects\<STM32xxx_yyy>\Templates_LL:

   a) Select an LL example To find the board on which LL examples are deployed, refer to the list of LL examples in STM32CubeProjectsList.html, to *Table 3: Number of examples available for each board*.

   b) Port the LL example

   – Copy/paste the Templates_LL folder to keep the initial source or directly update the existing *Templates_LL* project.

   – Then LL example porting consists mainly in replacing *Templates_LL* files by the *Examples_LL* targeted project.

   – 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:

   – Replace stm32f4xx_it.h file

   – Replace stm32f4xx_it.c file

   – Replace main.h file and update it: keep the LED and user button definition of the LL template under "BOARD SPECIFIC CONFIGURATION" tags.

Replace the main.c file, and update it: Keep the clock configuration of the SystemClock_Config() LL template: function under "BOARD SPECIFIC CONFIGURATION" tags.
Depending on the LED definition, replace all LEDx occurrences with another LEDy available in main.h.
Thanks to these adaptations, the example should be functional on the targeted board.

4.3 Using STM32CubeMX for generating the initialization C code

Another alternative to Steps 1 to 6 described in the *Section 4.2* consists in using the STM32CubeMX tool to easily generate code for the initialization of the system, the peripherals and middleware (Steps 1 to 6 above) 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 thanks to a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and an 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 “STM32CubeMX for STM32 configuration and initialization C code generation” user manual (UM1718).

4.4 How to get STM32CubeF4 release updates

The STM32CubeF4 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 the *www.st.com* website and proposes to download them to the user’s computer.

4.4.1 How to install and run the STM32CubeUpdater program

- Double-click SetupSTM32CubeUpdater.exe file 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).

5 FAQs

What is the license scheme for the STM32CubeF4 firmware?

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license.

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

The middleware based on the 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 firmware package?

The STM32CubeF4 firmware 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.

Is there any link with the 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.

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, 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, they are. The STM32CubeF4 provides a rich set of examples and applications (140 for STM324x9I-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, 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 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 STM32CubeF4 firmware releases?

The STM32CubeF4 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 STM32F4 configuration and initialization C code generation, the user can benefit from the STM32CubeMX self-updates as well as STM32CubeF4 firmware package updates.

For more details, refer to *Section 4.4*.

Is there any link with Standard Peripheral Libraries?

The STM32Cube HAL and LL drivers are the replacement of the standard peripheral library:

- The HAL drivers offer a higher abstraction level compared to the standard peripheral APIs. They focus on peripheral common 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 accesses. 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(s).

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. 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 Examples_MIX example.

Is there any LL APIs which are not available with HAL

Yes, there are.

A few Cortex® APIs have been added in 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 in the toolchain compiler preprocessor.

6 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 1: STM32CubeF4 main features*. Added LibJPEG in *Section 2: 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 firmware package structure*. Updated *Section 4.2: How to develop an application* and *Section 4.3: 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: Macros for STM32F4 series* and *Table 2: Evaluation and Discovery boards for STM32F4 series* and *Table 3: Number of examples and applications available for each board*. – Updated *Section 5: FAQs*.
19-May-2015	4	Added support for SW4STM32 toolchain.
21-Sep-2015	5	Added support for STM32F469xx STM32F410xx. Updated *Table 1: Macros for STM32F4 series* and *Table 2: Evaluation and Discovery boards for STM32F4 series*.
12-Nov-2015	6	Added support for STM32F411xE Updated *Table 1: Macros for STM32F4 series*,*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: Macros for STM32F4 series*, *Table 2: Evaluation and Discovery boards for STM32F4 series*, *Table 3: Number of examples and applications available for each board*.
02-Nov-2016	8	Added support for STM32F413xx and STM32F423xx. Updated *Table 1: Macros for STM32F4 series*, *Table 2: Evaluation and Discovery boards for STM32F4 series*, *Table 3: Number of examples and applications available for each board*.

Table 4. Document revision history (continued)

Date	Revision	Changes
16-Feb-2017	9	Added support for LL drivers on cover page. Added *Section 4.2.1: HAL application* and *Section 4.2.2: LL application* Updated: – *Figure 1: STM32CubeF4 firmware components* – *Figure 2: STM32CubeF4 firmware architecture* – *Figure 3: STM32CubeF4 firmware package structure* – *Figure 4: STM32CubeF4 examples overview* – *Table 3: Number of examples and applications available for each board*. – Added new FAQs in *Section 5: FAQs*.
17-Feb-2017	10	Corrected date for revision 9 in document revision history table.