UM1739
User manual

Getting started with STM32CubeF2 firmware package for STM32F2 Series

Introduction

The STMCube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube 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 STM32CubeF2 for STM32F2 Series)
  - The STM32Cube HAL, an STM32 abstraction-layer embedded software, ensuring maximized portability across STM32 portfolio
  - The 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 only 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.

Contents

List of tables

List of figures

1 STM32CubeF2 main features

STM32CubeF2 gathers in a single package all the generic embedded software components required to develop an application on STM32F2 microcontrollers. In line with the STM32Cube initiative, this set of components is highly portable, not only within the STM32F2 Series but also to other STM32 series.

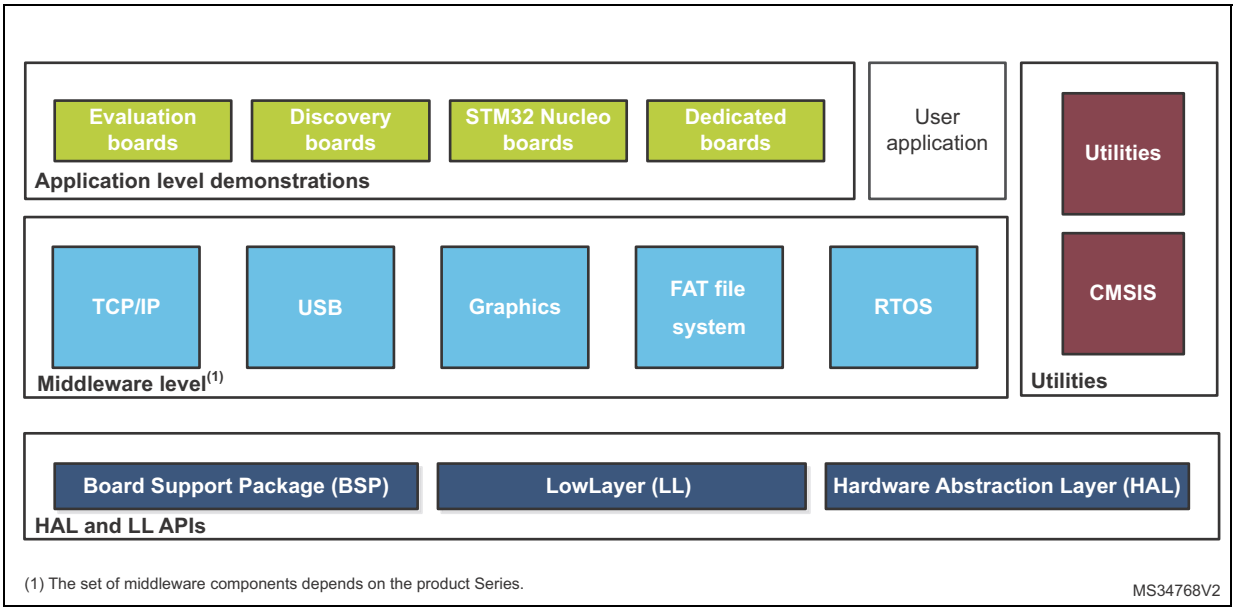
STM32CubeF2 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes the Low Layer (LL) and the Hardware Abstraction Layer (HAL) APIs that cover the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL and LL APIs are available in an open-source BSD license for user convenience.

The STM32CubeF2 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
- STemWin, a professional graphical stack solution available in binary format and based on the emWin solution from the ST partner SEGGER
- 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 PolarSSL

A demonstration implementing all these middleware components is also provided in the STM32CubeF2 package.

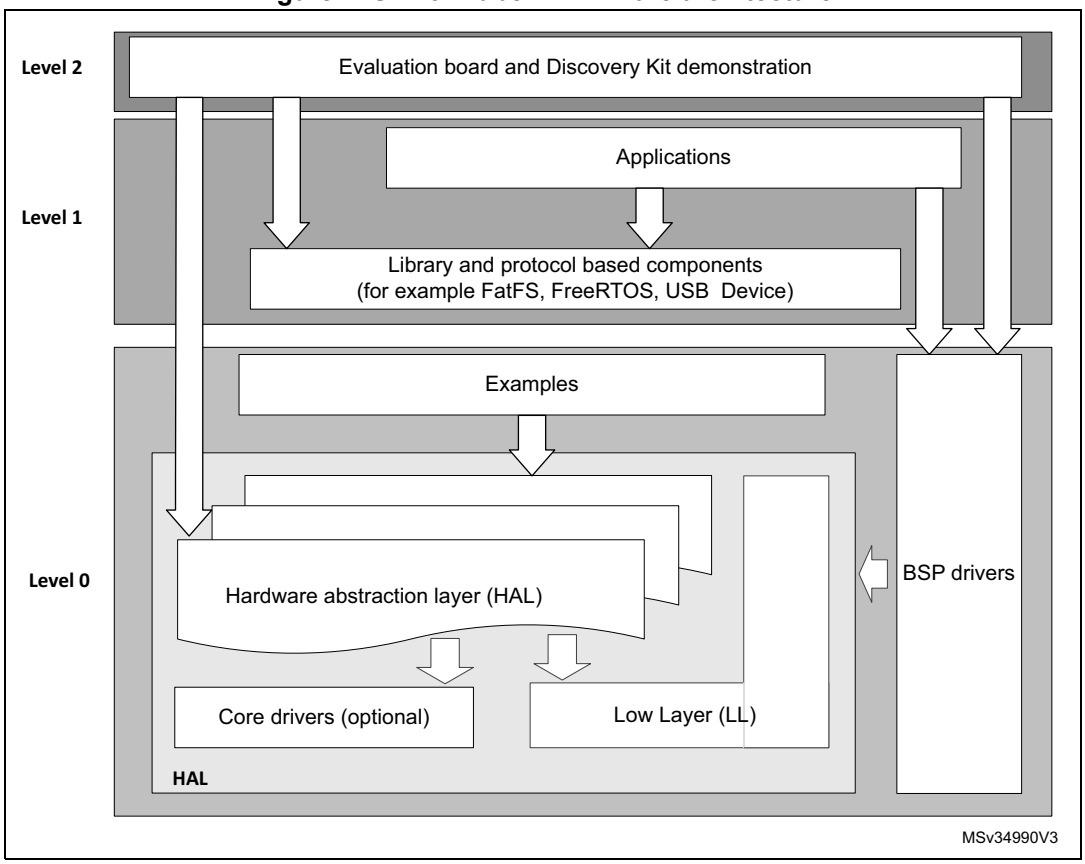
Figure 1. STM32CubeF2 firmware components



2 STM32CubeF2 architecture overview

The STM32Cube firmware solution is built around three independent levels that can easily interact with each other as described in *Figure 2*.

Figure 2. STM32CubeF2 firmware architecture



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 (for example audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers) and composed of two parts:
  - Component: this 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 other 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().

  The BSP 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 a 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 (I2S, UART…) it provides

APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and handle communication errors that may raise during communication. The HAL Drivers APIs are split into two categories:

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

- **Basic peripheral usage examples**: this layer contains the examples of the basic operation of the STM32F2 peripherals using either the HAL or/and the Low Layer drivers APIs as well as the BSP resources.
- **Low Layer (LL)**:
  - The low layer APs provide low-level APIs at register level, with a better optimization but less portability. They require a deep knowledge of MCU and peripheral specifications. The LL drivers are designed to offer a fast lightweight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, the LL APIs are not provided for peripherals where the optimized access is not a key feature, or those requiring a heavy software configuration and/or a complex upper-level stack (such as FMC, 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
  - A function for peripheral de-initialization (peripheral registers restored to their default values)
  - A set of inline functions for a direct and atomic register access
  - A full independence from the HAL and the capability to be used in standalone mode (without HAL drivers)
  - A full coverage of the supported peripheral features.

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

- **Middleware components**: set of Libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. The 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. As an example, the FatFs implements the disk I/O driver to access the microSD drive or the USB Mass Storage Class.
- **Examples based on the middleware components**: each middleware component comes with one or more examples (called also Applications) showing how to use the component. Integration examples using 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 basic peripheral usage applications for board-based functions.

In the current version of the STM32CubeF2 firmware package, no Level 2 projects are provided. The user can rely on the Level 2 projects provided with the STM32CubeF4 firmware package as example.

3 STM32CubeF2 firmware package overview

3.1 Supported STM32F2 Series devices and hardware

STM32Cube offers a highly portable Hardware Abstraction Layer (HAL) built around a generic and modular architecture. It allows the upper layers, the middleware and application, to implement its functions without knowing, in-depth, the MCU used. This improves the library code re-usability and guarantees an easy portability from one device to another.

The STM32CubeF2 offers a full support for all the STM32F2 Series devices. The user only needs to define the right macro in stm32f2xx.h.

Table 1 lists which macro to define depending on the used STM32F2 device. Note that the macro must also be defined in the compiler preprocessor.

Table 1. Macros for STM32F2 Series

Macro defined in stm32f2xx.h	STM32F2 devices
STM32F205xx	STM32F205RB,STM32F205RC, STM32F205RE, STM32F205RF, STM32F205RG, STM32F205VB, STM32F205VC, STM32F205VE, STM32F205VF STM32F205VG, STM32F205ZC, STM32F205ZE, STM32F205ZF and STM32F205ZG
STM32F215xx	STM32F215RE, STM32F215RG, STM32F215VE, STM32F215VG, STM32F215ZE and STM32F215ZG
STM32F207xx	STM32F207IC, STM32F207IE, STM32F207IF, STM32F207IG, STM32F207VC, STM32F207VE, STM32F207VF, STM32F207VG, STM32F207ZC, STM32F207ZE, STM32F207ZF and STM32F207ZG
STM32F217xx	STM32F217VG, STM32F217VE, STM32F217ZG, STM32F217ZE, STM32F217IG and STM32F217IE

STM32CubeF2 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 the STMicroelectronics boards listed in *Table 2*.

Table 2. STMicroelectronics boards for STM32F2 Series

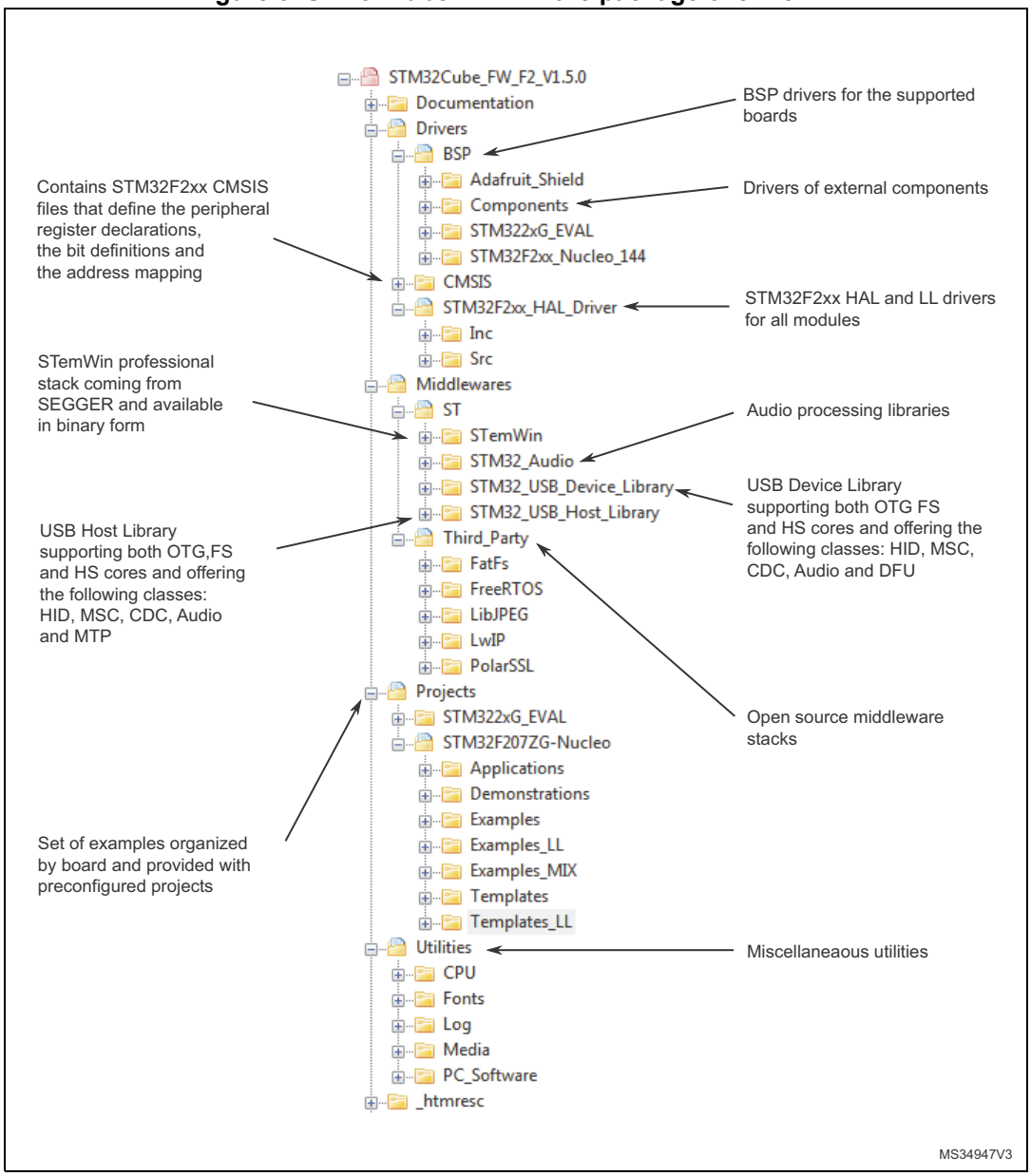
Board	STM32F2 devices supported
STM322xG_EVAL	STM32F207xx and STM32F217xx
STM32F207ZG-Nucleo	STM32F207ZG

The STM32CubeF2 firmware can run on any compatible hardware. Simply update the BSP drivers to port the provided examples on the user board if its hardware features are the same (for example LED, LCD Display, buttons).

3.2 Firmware package overview

The STM32CubeF2 firmware solution is provided in one single zip package with the structure shown in *Figure 3*.

Figure 3. STM32CubeF2 firmware package overview(1)

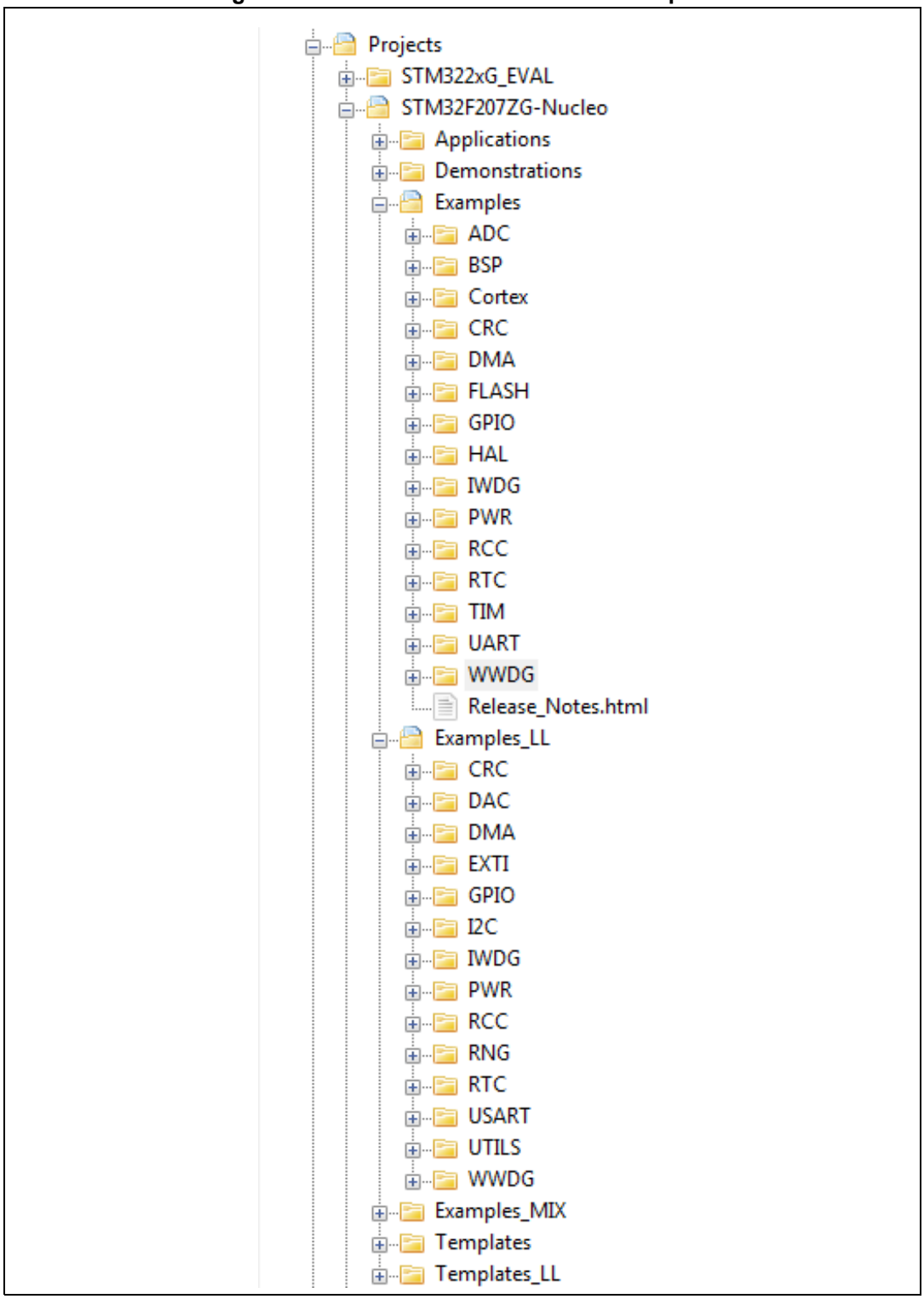


1. The library files cannot be modified by the user while the set of examples are modifiable files.

For each board supporting the devices of the STM32F2 Series, a set of examples with preconfigured projects is provided for EWARM, MDK-ARM, TrueSTUDIO and SW4STM32 toolchains.

Figure 4 shows the projects structure for the STM32F207ZG-Nucleo board.

Figure 4. Overview of STM32CubeF2 examples



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

- Examples in level 0 are called Examples, Examples_LL, and Examples_MIX. They use respectively HAL drivers, LL drivers and 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.

The *Template* project is provided to allow to build quickly any firmware application on a given board.

All examples have the same following 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 required environment.

Table 3 provides the number of projects available for each board.

Table 3. Number of examples available on the boards

Board	Examples	Examples_ LL	Examples_ MIX	Applications	Demonstration
STM322xG_EVAL	75	NA	NA	50	0
STM32F207ZG-Nucleo	25	69	12	7	1

4 Getting started with STM32CubeF2

4.1 How to run a first example

This section explains how to run a first example within STM32CubeF2, using as illustration the generation of a simple led toggle on STM322xG_EVAL board:

1. Download the STM32CubeF2 FW package
2. Unzip the package into a selected directory. Ensure that the package structure is not modified (as shown in the *Figure 3*). It is also recommended to copy the package at a location close to the root volume (for example *C:\Eval* or *G:\Tests*) because some IDEs encounter problems when the path length is too high.

- Browse to \Projects\STM322xG_EVAL\Examples
- Open *GPIO* folder, then the *GPIO_EXTI* folder
- Open the project with the preferred toolchain (see the examples below)
- Rebuild all files and load the image into the target memory
- Run the example: the behavior of the program is described in the *readmed.txt* file available with the example (for example the usage of User Button or the meaning of the LEDs).

Below is 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[a]
  - Rebuild all files: **Project > Rebuild all**
  - Load the project image: **Project > Debug**
  - Run program: **Debug > Go** (F5)
- MDK-ARM
  - Under the example folder, open *\MDK-ARM* subfolder
  - Launch the *Project.uvproj* workspace[a]
  - Rebuild all files: **Project > Rebuild all target files**
  - Load the project image: **Debug > Start/Stop Debug Sessio**n
  - Run the program: **Debug > Run** (F5)
- TrueSTUDIO
  - Open the TrueSTUDIO toolchain
  - Select **File > Switch Workspace > Other** and browse to TrueSTUDIO workspace directory
  - Select **File > Import**, select **General > Existing Projects into Workspace** and then select **Next**.
  - Browse to the TrueSTUDIO workspace directory, and select the **project**
  - Rebuild all project files: select the project in the **Project explorer** window then select **Project > Build project menu**.
  - Run the program: **Run > Debug** (F11)
- SW4STM32
  - Open the SW4STM32 toolchain
  - Click on **File > Switch Workspace->Other** and browse to the SW4STM32 workspace directory
  - Click on **File > Import, select General > 'Existing Projects into Workspace'** and then select **Next**.
  - Browse to the SW4STM32 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)

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

4.2 How to develop an application

4.2.1 HAL application

This section describes the successive steps to create an application using STM32CubeF2.

1. **Create a project:** to create a new project, start either 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.
   Note: <STM32xx_xxx> refers to the board name, for example STM322xG_EVAL.

   The Template project provides an empty main loop function. It is a good starting point to get familiar with the project settings for STM32CubeF2. It has the following characteristics:

   a) It contains the sources of HAL, CMSIS and BSP drivers which are the minimum required components to develop a code for a given board.

   b) It contains the include paths for all the firmware components.

   c) It defines the STM32F2 device supported, allowing to configure the CMSIS and HAL drivers accordingly.

   d) It provides ready-to use user files preconfigured as, for example:

   - HAL is initialized.
   - SysTick ISR implemented for HAL_Delay() purpose.
   - System clock is configured with the maximum frequency of the device

Note: When copying an existing project to another location, make sure to update the include paths.

2. **Add the necessary middleware to the project (optional):** the available middleware stacks are USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. To find out which source files must be added to the project files list, refer to the documentation provided for each middleware.
   To get a better view of the sources files to be added and the include paths, look at the **Applications** available under \Projects\STM32xx_xxx\Applications\<MW_Stack>, where <MW_Stack> refers to the middleware stack, for example USB_Device.
3. **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.*
   Make sure to remove the word “*_template*” when copying the file 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.

4. **Start the HAL Library**: after jumping to the main program, the application code calls the *HAL_Init()* API to initialize the HAL library, which does the following:
   a) Configure the Flash prefetch, instruction and data caches (configured by the user through macros defined in *stm32f2xx_hal_conf.h*).
   b) Configure the Systick to generate an interrupt each 1 msec, 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 16 MHz).
   c) Set NVIC Group Priority to 4.
   d) Call HAL_MspInit() callback function defined in the user file *stm32f2xx_hal_msp.c t*o set the global low-level hardware initializations

5. **Configure the system clock:** the system clock configuration is done by calling the two following APIs
   – HAL_RCC_OscConfig(): configures the internal and/or external oscillators, the PLL source and factors. The user may select to configure one oscillator or all oscillators. In addition, the PLL configuration can be skipped if there is no need to run the system at high frequency.
   – 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 *stm32f2xx_it.c* to call the required interrupt handlers (peripheral and DMA), if needed.
   c) Implement the callback functions (these functions are called when the peripheral process is complete or/and when an error occurs), if the user plans to use the 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.
   – The HAL provides intuitive and ready-to-use APIs to configure the peripheral. The HAL supports polling, IT and DMA programming model, to accommodate any application requirements. For more details on how to use each peripheral, refer to the set of examples available in the firmware package.
   – If the application has some real-time constraints, the user can find a set of examples showing how to use the FreeRTOS and integrate it with all middleware stacks provided within STM32CubeF2. It can be a good starting point for a first development.
   – **Important 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 an LL application using STM32CubeF2.

1. **Create a project**

   To create a new project, start either 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> which refers to the board name, such as STM32F207ZG-Nucleo).

   The *Template* project provides an empty main loop function, however it is a good starting point to get familiar with project settings for STM32CubeF2.

   The template main characteristics are listed below:

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

   b) It contains the include paths for all the required firmware components.

   c) It selects the supported STM32F2 device and allows configuring the CMSIS and LL drivers accordingly.

   d) 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 configured with the 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 on the boards*.

   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 the *Templates_LL* files by the *Examples_LL* targeted.

   – Keep all board specific parts. For reasons of clarity, the 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 stm32f2xx_it.h file

   – Replace stm32f2xx_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 main.c file, and update it:
  - Keep the clock configuration of the SystemClock_Config() LL template: function under "BOARD SPECIFIC CONFIGURATION" tags.
  - Depending on LED definition, replace all LEDx_PIN by another LEDx (number) available in main.h file.

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

4.3 Using STM32CubeMX to generate the initialization C code

Another alternative to steps 1 to 6 described in *Section 4.2: How to develop an application* 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. Select the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
2. Configure each required embedded software using a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and the utility performing MCU peripheral configuration (for example GPIO and USART) and middleware stacks (for example USB, TCP/IP).
3. Generate 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 Getting STM32CubeF2 release updates

The STM32CubeF2 firmware package comes with an updater utility: the STM32CubeUpdater, also available as a menu within STM32CubeMX code generation tool. The updater solution detects new firmware releases and patches available on *www.st.com* and proposes to download them to the user's computer.

4.4.1 Installing and running 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. For more details on the Updater configuration, refer to *Section 3* of the STM32CubeMX user manual (UM1718).

5 Frequently asked questions (FAQs)

What is the license scheme for the STM32CubeF2 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 well-known open-source solutions (FreeRTOS, FatFs, LwIP and PolarSSL) have user-friendly license terms. For more details, refer to the license agreement of each Middleware.

Which boards are supported by the STM32CubeF2 firmware package?

The STM32CubeF2 firmware package provides BSP drivers and ready-to-use examples for the following STM32F2 boards: STM3220G_EVAL, STM3221G_EVAL and STM32F207ZG-Nucleo.

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. Existing Standard Peripheral Libraries are 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. STM32CubeF2 provides a rich set of examples and applications. 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 the user configuration.

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 defining 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. The 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 the HAL drivers, these LL initialization APIs allows 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?

The HAL drivers offer high-level and function-oriented APIs, with a high level of portability. the product/IPs complexity is hidden for end users.

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

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 the HAL and then manage the I/O operations with the LL drivers.

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

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

There is no configuration file. The source code shall directly include the necessary stm32f2xx_ll_ppp.h file(s).

Is there any LL APIs which are not available with HAL?

Yes, there are.

A few Cortex® APIs have been added in stm32f2xx_ll_cortex.h e.g. for accessing the SCB or the SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using the LL drivers in standalone mode, the user does not need to enable SysTick interrupts because they are not used in the LL APIs, while the HAL functions requires 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 the LL APIs, add this switch in the toolchain compiler preprocessor.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
02-Apr-2014	1	Initial release.
09-Sep-2015	2	Updated: – *Table 2: STMicroelectronics boards for STM32F2 Series*, – *Table 3: Number of examples available on the boards*, – *Section 3.2: Firmware package overview* with references to W4STM32 toolchain, – *Figure 4: Overview of STM32CubeF2 examples*, – *Section 4.1: How to run a first example* with the addition of SW4STM32, – *Section 4.2: How to develop an application*with the addition of item 6 “Peripheral initialization”. Added: – *Section 4.3: Using STM32CubeMX to generate the initialization C code*, – *Section 5: Frequently asked questions (FAQs)*.
19-Nov-2015	3	Updated: – *Table 2: STMicroelectronics boards for STM32F2 Series* adding a new row for STM32F207ZG-Nucleo board. – *Table 3: Number of examples available on the boards* adding a new row for STM32F207ZG-Nucleo board. – *Section 4.2: How to develop an application*changing the important note. – *Section 5: Frequently asked questions (FAQs)* 2nd question about the boards supported by STM32CubeF2 firmware package, adding STM32F207ZG-Nucleo board.
02-Mar-2017	4	Added low layer API (LL) feature: – Updated cover – Updated *Section 1: STM32CubeF2 main features*. – Updated *Figure 1: STM32CubeF2 firmware components*. – Updated *Section 2: STM32CubeF2 architecture overview*. – Updated *Figure 2: STM32CubeF2 firmware architecture*. – Updated *Figure 3: STM32CubeF2 firmware package overview(1)*. Updated *Figure 4: Overview of STM32CubeF2 examples* for the STM32F207ZG-Nucleo board. Updated *Table 3: Number of examples available on the boards* adding Examples_LL and Examples_MIX columns for STM32F207ZG-Nucleo board. Updated *Section 4: Getting started with STM32CubeF2* adding *Section 4.2.2: LL application*. Updated *Section 5: Frequently asked questions (FAQs)*.