UM1739
User manual

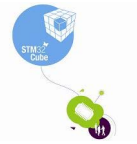
Getting started with STM32CubeF2 firmware package for STM32F2 Series

Introduction

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,
  - A consistent set of middleware components such as RTOS, USB, TCP/IP and graphics,
  - All embedded software utilities coming with a full set of examples.



Contents

List of tables

List of figures

1 STM32CubeF2 main features

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

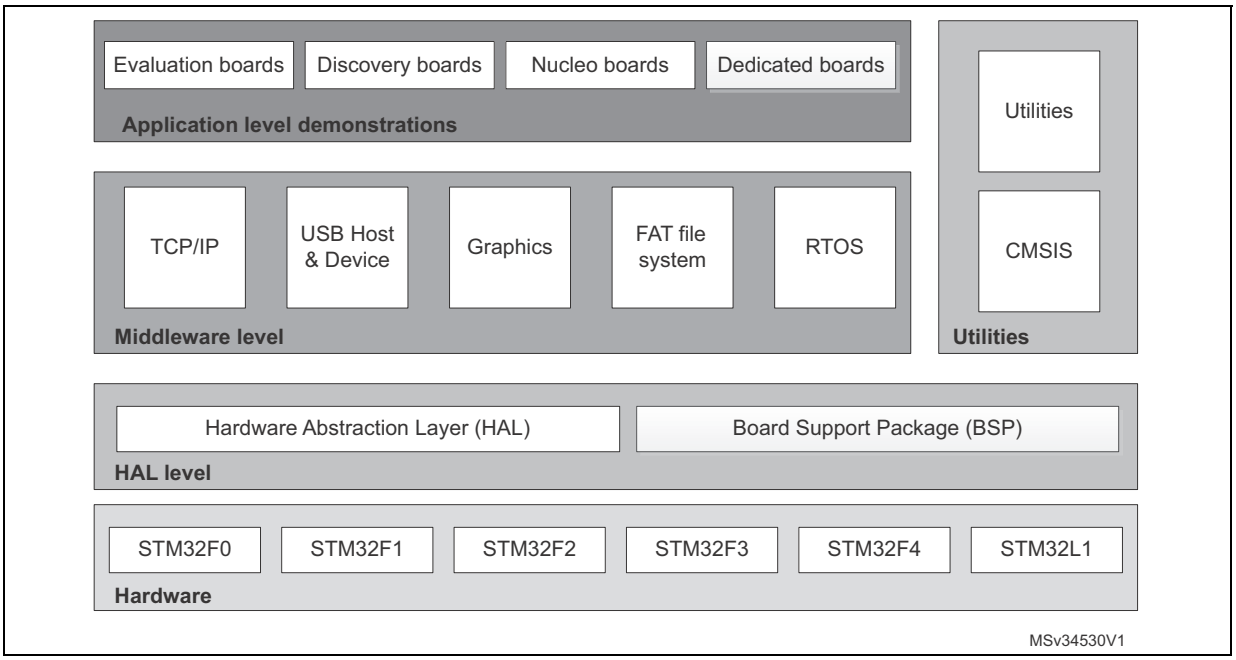
STM32CubeF2 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes a low-level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for user convenience.

STM32CubeF2 package also contains a set of middleware components with the corresponding examples. They come in 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 ST partner solution SEGGER emWin
- 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.

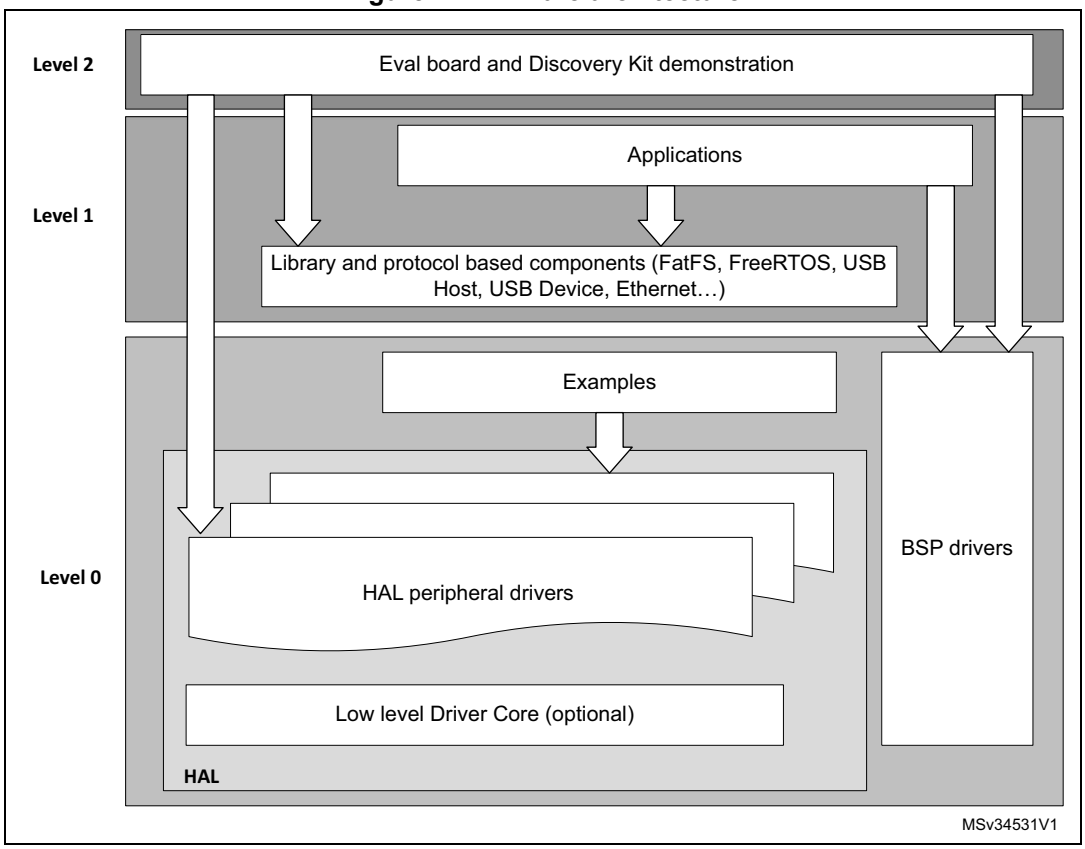
Figure 1. STM32CubeF2 firmware components



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. 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 in the hardware boards (for example Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers) and composed of two parts:
  - Component: the driver related to the external device on the board and not related to the STM32, the component driver provide specific APIs to the BSP driver external components and can be ported on 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().

  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. As an example, for the communication peripherals (I2S, UART…) it

provides APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and manage communication errors that may raise during communication. The HAL Drivers APIs are split 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 examples of basic operation of the STM32F2 peripherals using only the HAL and BSP resources.

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 (also called Applications) showing how to use the component. Integration examples using several Middleware components are also provided.

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. You can rely on the Level 2 projects provided with the STM32CubeF4 firmware package as example.

3 STM32CubeF2 firmware package overview

3.1 Supported STM32F2 devices and hardware

STM32Cube offers a highly portable Hardware Abstraction Layer (HAL) built around a generic architecture allowing the upper layers, for example the Middleware layer, to implement its functions without the need to know in-depth the MCU in use. This improves the library code re-usability and guarantees an easy portability from one device to another.

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

Table 1 lists the macro to define depending on the STM32F2 device in use. 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

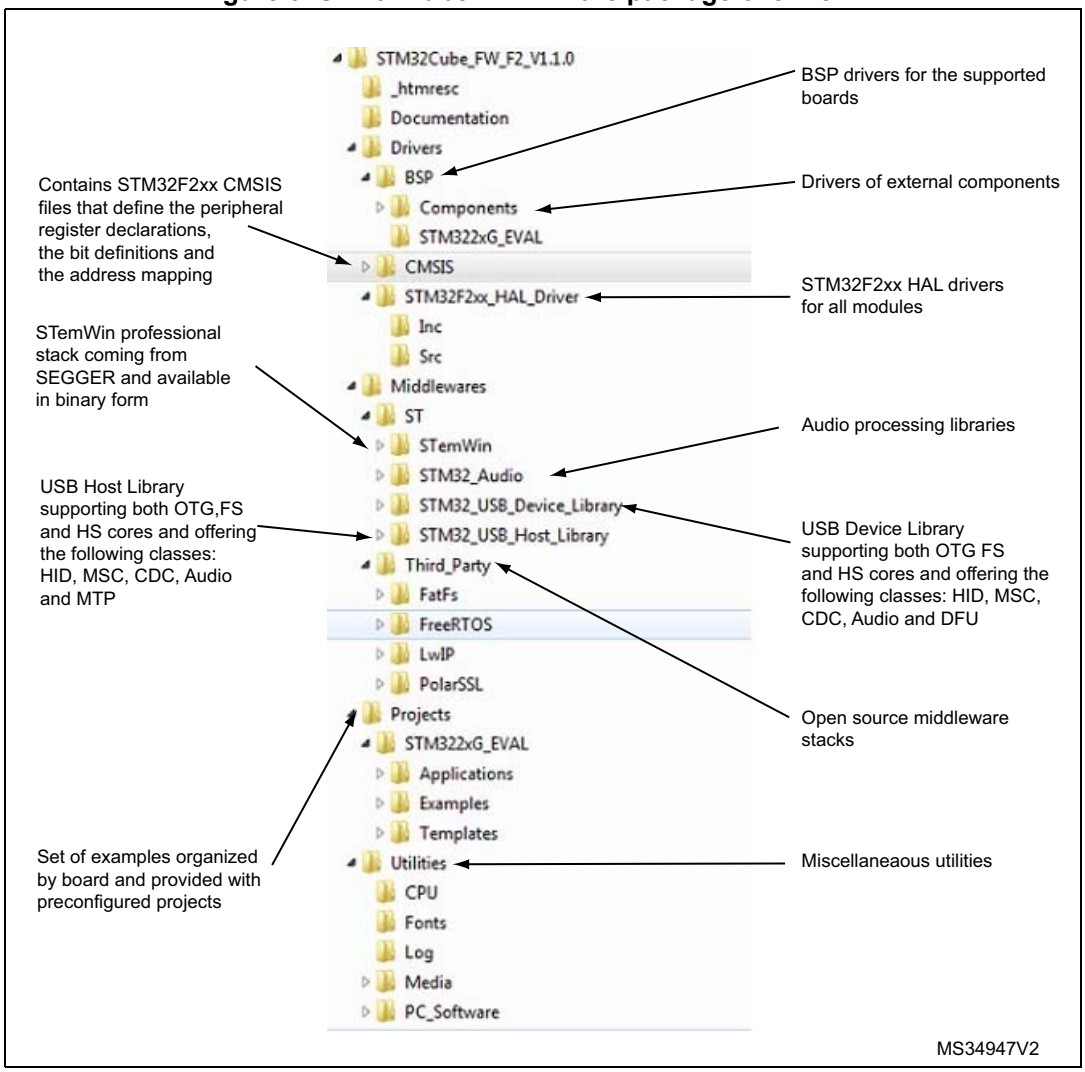
Board	STM32F2 devices supported
STM322xG_EVAL	STM32F207xx and STM32F217xx
STM32F207ZG-Nucleo	STM32F207ZG

The STM32CubeF2 firmware is able to run on any compatible hardware. The users can simply update the BSP drivers to port the provided examples on their own board, providing that their board support the same hardware functions (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* hereafter.

Figure 3. STM32CubeF2 firmware package overview (1)

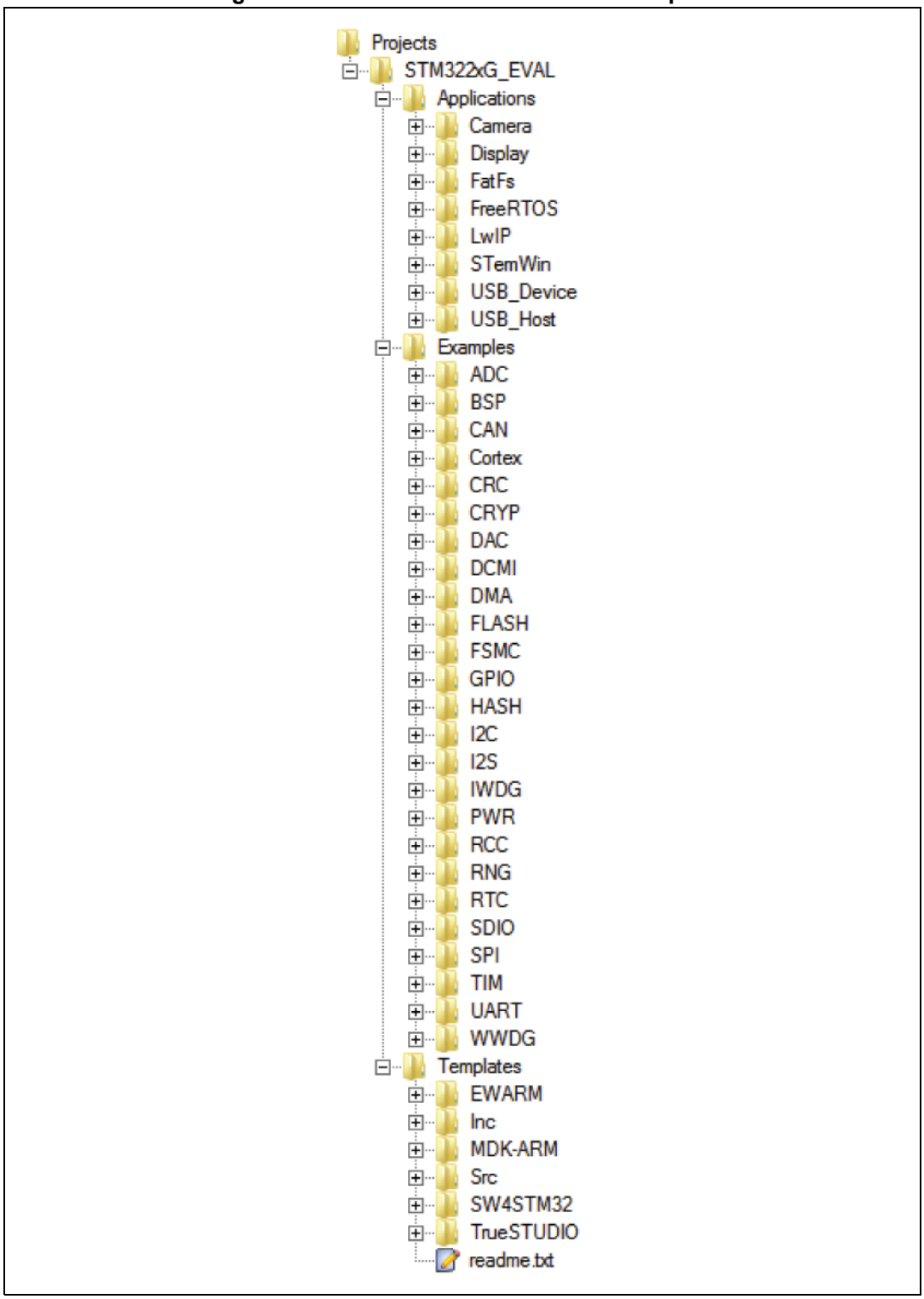


1. The library files cannot be modified by the user while the set of examples are modifiable files.

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 STM322xG-EVAL board.

Figure 4. Overview of STM32CubeF2 examples



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

- Examples in level 0 are called *Examples*. They use HAL drivers without any Middleware component.
- Examples in level 1 are called *Applications*. They provide typical use cases of each Middleware component.

The *Template* project available in the Template directory is provided for the users to quickly build 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 containing the preconfigured project for each toolchain,
- *readme.txt* describing 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	Applications	Demonstration
STM322xG_EVAL	75	50	0
STM32F207ZG-Nucleo	25	7	1

4 Getting started

4.1 Running your 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 directory of your choice. Make sure not to modify the package structure shown in the *Figure 3*. It is also recommended to copy the package at a location close to your 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, and then open *GPIO_EXTI* folder
- Open the project with your preferred toolchain (see the examples below)
- Rebuild all files and load your 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 Developing your own application

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

1. **Creating your project:** to create a new project you can either 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.
   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 on 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: If you copy an existing project to another location make sure to update the include paths.

2. **Adding the Middleware to your project (optional):** the available Middleware stacks are USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. To know which source files you need to add in the project files list refer to the documentation provided for each Middleware.
   For 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. **Configuring 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. **Starting the HAL Library**: after jumping to the main program, the application code calls the *HAL_Init()* API to initialize the HAL library, and do 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. **Configuring 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 can choose to configure one oscillator or all oscillators. In addition, the user can choose to skip the PLL configuration 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 prescaler.
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 you plan 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. **Developing your application process:** at this stage, your system is ready and you can start developing your 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 your application has some real time constraints, use the set of examples showing how to use FreeRTOS and its integration with all Middleware stacks provided within STM32CubeF2. This is a good starting point for your 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 please refer to HAL_TimeBase example.

4.3 Using STM32CubeMX to generate the initialization C code

Another alternative to steps 1 to 6 in *Section 4.2: Developing your own 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. Selection of the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
2. Configuration of 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. 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, please refer to the section *STM32CubeMX for STM32 configuration and initialization C code generation* in the STM32CubeMX user guide user (UM1718).

4.4 Accessing 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 from *www.st.com* and offers to download these on 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 is displayed 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 *Section 3* in STM32CubeMX user guide (UM1718) for more details on the updater configuration).

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 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.

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: Running your first example* with the addition of SW4STM32, – *Section 4.2: Developing your own 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: Developing your own 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.