Getting started with STM32CubeL1 MCU Package for STM32L1 Series

Introduction

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

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of the C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL1 for STM32L1 Series):
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio.
  - The Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL.LL APIs are available only for a set of peripherals
  - A consistent set of middleware components such as RTOS, USB, STMTouch, FatFS, and graphics
  - All embedded software utilities, delivered with a full set of examples.

This user manual describes how to get started with the STM32CubeL1 MCU Package.

The present user manual describes the main features of the STM32CubeL1 MCU Package, and then provides an overview of the STM32CubeL1 architecture and Package structure.

Section 2 and Section 3 provide an overview of the STM32CubeL1 architecture and MCU Package structure.

1 STM32CubeL1 main features

The STM32CubeL1 MCU Package runs on STM32 32-bit microcontrollers based on the Arm® Cortex® -M processor.

STM32CubeL1 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32L1 microcontrollers. In line with the STM32Cube initiative, this set of components is highly portable, not only within the STM32L1 but also to other STM32 series.

STM32CubeL1 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.

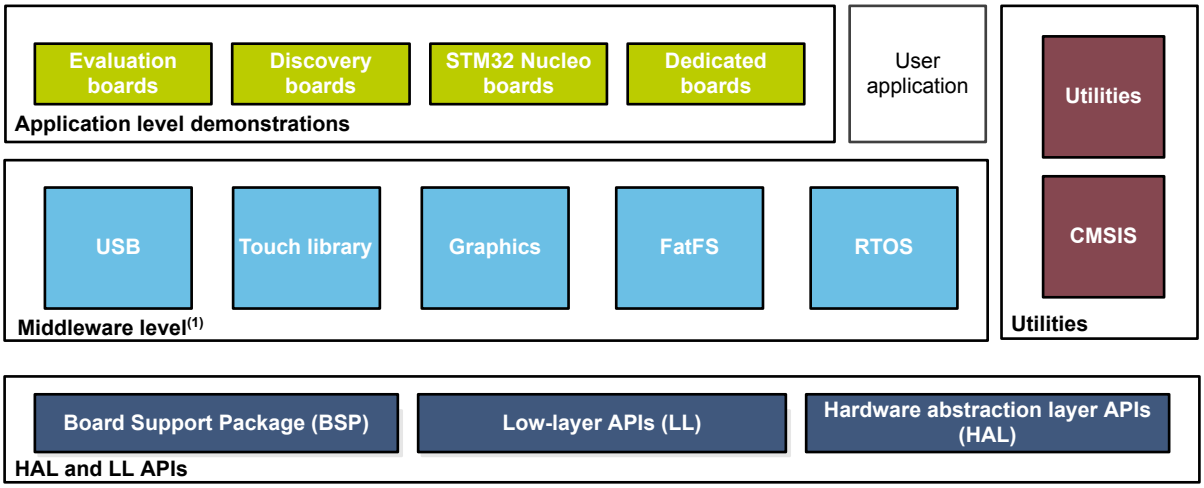
The STM32CubeL1 MCU Package also contains a set of middleware components with the corresponding examples. They come with very permissive license terms:

- Full USB device stack supporting many classes
  - Device classes: Audio, HID, MSC, CDC, DFU
- CMSIS-RTOS implementation with FreeRTOS™ open source solution
- FAT file system based on open source FatFS solution
- STMTouch touch sensing solutions
- STemWin, a professional graphical stack solution available in binary format and basedon ST partner solution SEGGER emWin.

A demonstration implementing all these middleware components is also provided in the STM32CubeL1 MCU Package.

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

Figure 1. STM32CubeL1 firmware components

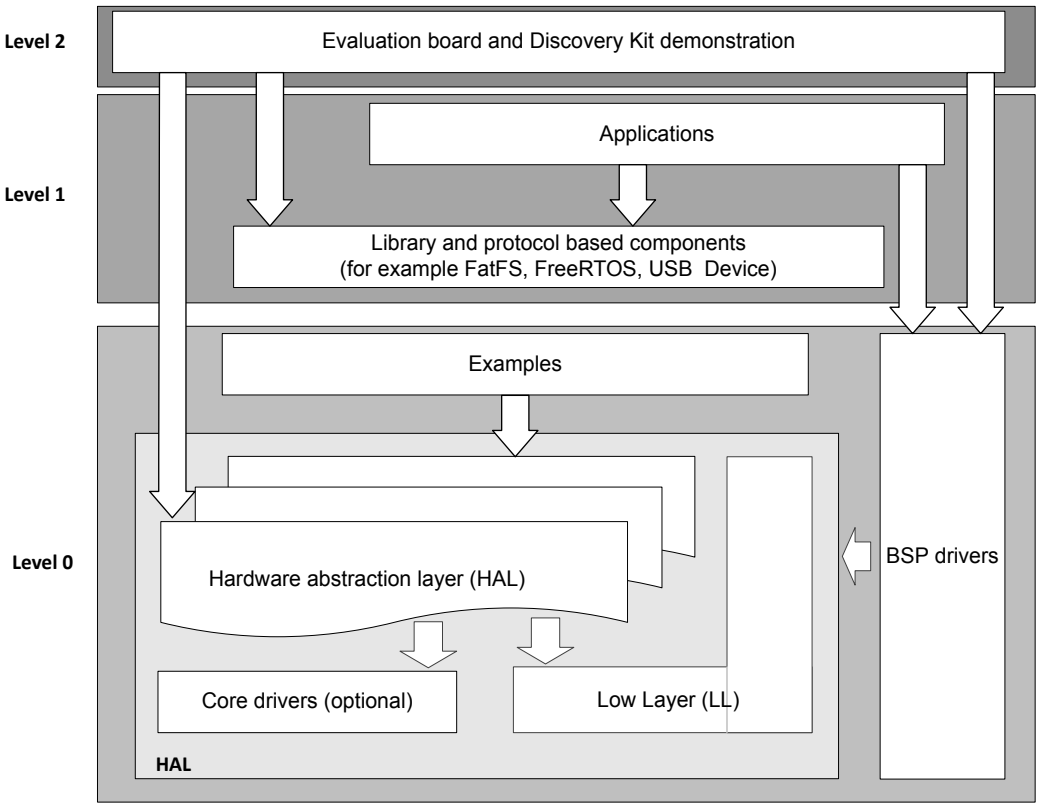


(1) The set of middleware components depends on the product Series.

2 STM32CubeL1 architecture overview

The STM32CubeL1 firmware solution is built around three independent levels that can easily interact with each other as described in the following figure.

Figure 2. STM32CubeL1 firmware architecture



2.1 Level 0

This level is divided into three sub-layers:

- Board support package (BSP)
- Hardware abstraction layer (HAL)
  - HAL peripheral drivers
  - Low-layer drivers
- Basic peripheral usage examples

2.1.1 Board support package (BSP)

This layer offers a set of APIs related to the hardware components on the hardware boards (such as LCD drivers, microSD™) and composed of two parts:

- Component

This is the driver relative to the external device on the board and not 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 allows linking the component driver to a specific board and provides a set of user-friendly APIs. The API naming rule is BSP_FUNCT_Action().

Examples: BSP_LED_Init(), BSP_LED_On()

The BSP is based on a modular architecture allowing an easy porting on any hardware by just implementing the low level routines.

2.1.2 Hardware abstraction layer (HAL) and low layer (LL)

The STM32CubeL1 HAL and LL are complementary and cover a wide range of applications requirements:

- The HAL drivers offer high-level function-oriented highly-portable APIs. They hide the MCU and peripheral complexity to end user.

  The HAL drivers provide generic multi-instance feature-oriented APIs which simplify user application implementation by providing ready to use process. As example, for the communication peripherals (I2S, UART…), it provides APIs allowing initializing and configuring the peripheral, managing data transfer based on polling, interrupt or DMA process, and handling communication errors that may raise during communication.

  The HAL driver APIs are split in two categories:

  – Generic APIs which provides common and generic functions to all the STM32 series
  – Extension APIs which provides specific and customized functions for a specific family or a specific part number.

- The low-layer APIs provide low-level APIs at register level, with better optimization but less portability. They require a deep knowledge of MCU and peripheral specifications. 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 for those requiring heavy software configuration and/or complex upper-level stack (such as USB).

  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 direct and atomic register access
  – A full independence from HAL and capability to be used in standalone mode (without HAL drivers)
  – A full coverage of the supported peripheral features.

2.1.3 Basic peripheral usage examples

This layer includes the examples build over the STM32 peripheral and using either the HAL or/and the low-layer drivers APIs as well as the BSP resources.

2.2 Level 1

This level is divided into two sub-layers:

- Middleware components
- Examples based on the middleware components.

2.2.1 Middleware components

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

The main features of each middleware component are as follows:

- **USB device libraries**
  – Several USB classes supported (Mass-Storage, HID, CDC, DFU, AUDIO, MTP).
  – Support of multi-packet transfer features that allows sending big amounts of data without splitting them into maximum packet size transfers.

  - Use of configuration files to change the core and the library configuration without changing the library code (Read Only).
  - RTOS and Standalone operation.
  - Link with low-level driver through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level drivers.
- **STemWin Graphical stack**
  - Professional grade solution for GUI development based on SEGGER's emWin solution
  - Optimized display drivers.
  - Software tools for code generation and bitmap editing (STemWin Builder…).
- **FreeRTOS™**
  - Open source standard.
  - CMSIS compatibility layer.
  - Tickless operation during low-power mode.
  - Integration with all STM32Cube middleware modules.
- **FAT file system**
  - FatFS FAT open source library.
  - Long file name support.
  - Dynamic multi-drive support.
  - RTOS and standalone operation.
  - Examples with microSD™.
- **STM32 touch sensing library**
  - Robust STMTouch capacitive touch sensing solution supporting proximity, touchkey, linear and rotary touch sensors. It is based a proven surface charge transfer acquisition principle.

2.2.2 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.

2.3 Level 2

This level is composed of a single layer which consist in 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 features.

3 STM32CubeL1 MCU Package overview

3.1 Supported STM32L1 devices and hardware

STM32Cube offers a highly portable hardware abstraction layer (HAL) built around a generic and modular architecture. It allows the upper layers, such as the middleware and application layers, to implement their functions without knowing, in-depth, the MCU used. This improves the library code re-usability and guarantees an easy portability on other devices.

The STM32CubeL1 offers full support for all STM32L1 devices. The user has only to define the right macro in stm32l1xxxx.h.

The following table gives the macro to define depending on the STM32L1 device used. This macro must also be defined in the compiler preprocessor.

Table 1. Macros for STM32L1 devices

Macro defined in stm32l1xx.h	STM32L1 devices
STM32L100xB	STM32L100C6, STM32L100R, STM32L100RB
STM32L100xBA	STM32L100C6-A, STM32L100R8-A, STM32L100RB-A
STM32L100xC	STM32L100RC
STM32L151xB	STM32L151C6, STM32L151R6, STM32L151C8, STM32L151R8,STM32L151V8, STM32L151CB, STM32L151RB, STM32L151VB
STM32L151xBA	STM32L151C6-A, STM32L151R6-A, STM32L151C8-A, STM32L151R8-A,STM32L151V8-A, STM32L151CB-A, STM32L151RB-A, STM32L151VB-A
STM32L151xC	STM32L151CC, STM32L151UC, STM32L151RC, STM32L151VC
STM32L151xCA	STM32L151RC-A, STM32L151VC-A, STM32L151QC, STM32L151ZC
STM32L151xD	STM32L151QD, STM32L151RD, STM32L151VD, STM32L151ZD
STM32L151xE	STM32L151QE, STM32L151RE, STM32L151VE, STM32L151ZE
STM32L151xDX	STM32L151VD-X
STM32L152xB	STM32L152C6, STM32L152R6, STM32L152C8, STM32L152R8, STM32L152V8, STM32L152CB, STM32L152RB, STM32L152VB
STM32L152xBA	STM32L152C6-A, STM32L152R6-A, STM32L152C8-A, STM32L152R8-A,STM32L152V8-A, STM32L152CB-A, STM32L152RB-A, STM32L152VB-A
STM32L152xC	STM32L152CC, STM32L152UC, STM32L152RC, STM32L152VC
STM32L152xCA	STM32L152RC-A, STM32L152VC-A, STM32L152QC, STM32L152ZC
STM32L152xD	STM32L152QD, STM32L152RD, STM32L152VD, STM32L152ZD
STM32L152xE	STM32L152QE, STM32L152RE, STM32L152VE, STM32L152ZE
STM32L152xDX	STM32L152VD-X
STM32L162xC	STM32L162RC, STM32L162VC
STM32L162xCA	STM32L162RC-A, STM32L162VC-A, STM32L162QC, STM32L162ZC
STM32L162xD	STM32L162QD, STM32L162RD, STM32L162VD, STM32L162ZD
STM32L162xE	STM32L162RE, STM32L162VE, STM32L162ZE
STM32L162xDX	STM32L162VD-X

The STM32CubeL1 features a rich set of examples and applications making it easy to understand and use any HAL driver and/or Middleware components. These examples are running on STMicroelectronics boards as listed in the following table:

Table 2. STM32 boards for STM32L1 devices

Board	STM32L1 devices supported
STM32L152D-EVAL	STM32L152xD
32L152CDISCOVERY	STM32L152xC
32L100CDISCOVERY	STM32L100xC
NUCLEO-L152RE	STM32L152xE

The STM32CubeL1 family supports Nucleo-64 boards.

- Nucleo-64 boards support Adafruit LCD display Arduino™ UNO shields which embedded 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.

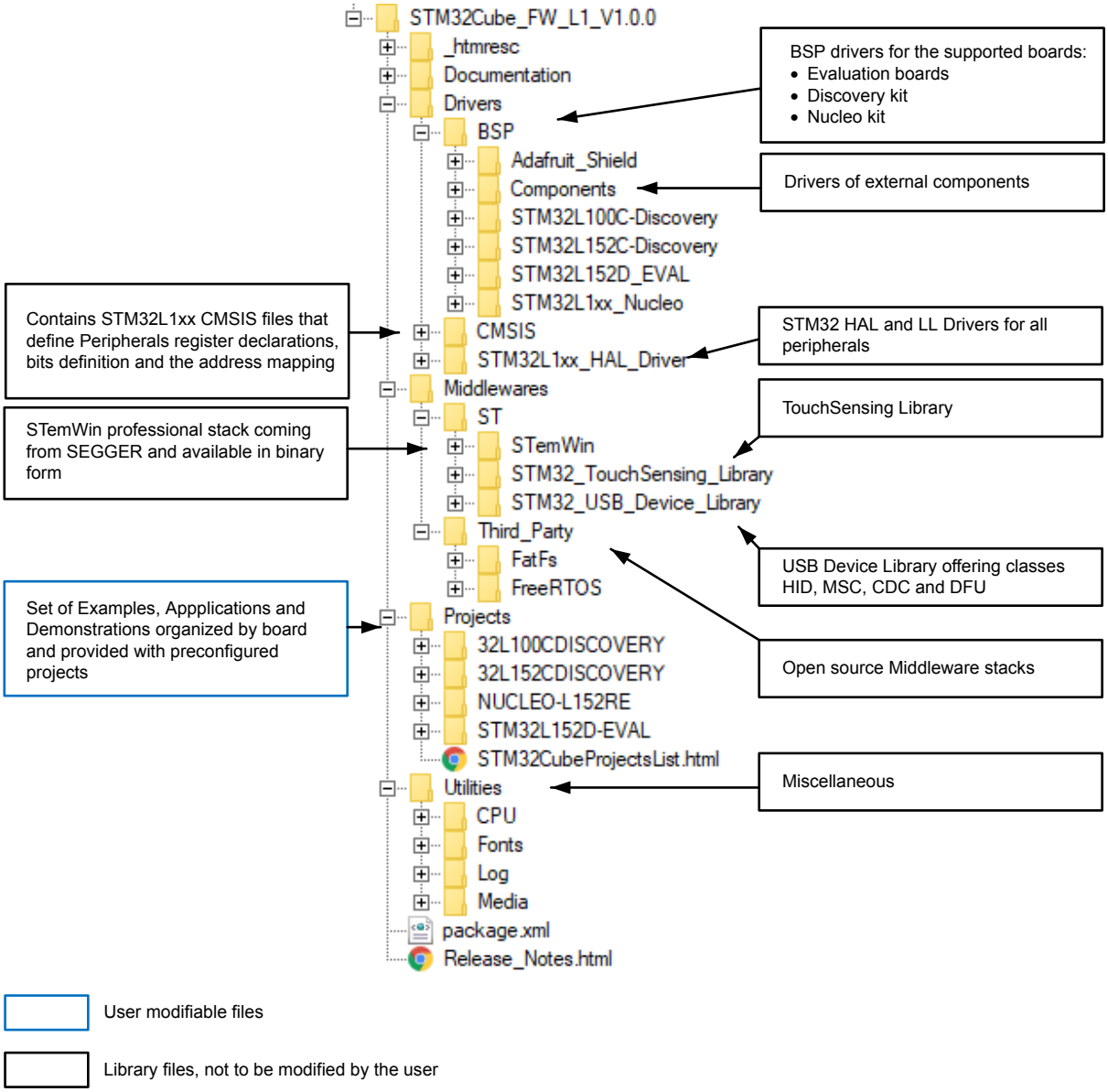
In the BSP component the dedicated drivers, for that Arduino™ shield are available and their use is illustrated through either the provided BSP example or in the Demonstration firmware, without forgetting the FatFS middleware application.

The STM32CubeL1 MCU Package is able to run on any compatible hardware. If the user's board has the same hardware features as the ST board (LED, pushbuttons and others), the user has just to update the BSP drivers to port the provided examples on his board.

3.2 MCU Package overview

The STM32CubeL1 firmware solution is provided in one single zip package having the structure shown in the following figure.

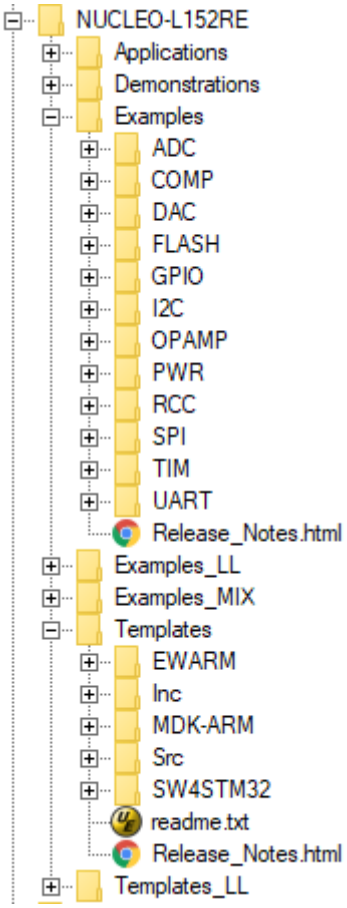
Figure 3. STM32CubeL1 MCU Package structure



For each board, a set of examples are provided with pre-configured projects for EWARM, MDK-ARM and SW4STM32 toolchains.

the following figure shows the project structure for the NUCLEO-L152RE board. The structure is identical for any other additional supported board.

Figure 4. Projects for NUCLEO-L152RE board



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, that provide typical use cases of each Middleware component
- Examples in level 2 are called Demonstration, that implement all the HAL, BSP and Middleware components

The template project available in the "Templates" and "Templates_LL" directories allows to quickly build any firmware application on a given board.

All examples have the same structure,

- \Inc folder that contains all header files
- \Src folder for the sources code
- \EWARM, \MDK-ARM and \SW4STM32 folders containing the pre-configured project for each toolchain
- readme.txt describing the example behavior and the required environment to make it work

The following table gives the number of projects available for each board.

Table 3. **Number of examples for each board**

Level	32L152CDISCOVERY	32L100CDISCOVERY	STM32L152D-EVAL	NUCLEO-L152RE	Total
Templates_LL	1	1	1	1	4
Templates	1	1	1	1	4
Examples_MIX	0	0	0	12	12
Examples_LL	0	0	0	73	73
Examples	7	7	28	27	69
Demonstrations	0	1	0	1	2
Applications	1	1	19	3	24
Total	10	11	49	118	188

4 Getting started with STM32CubeL1

4.1 Running your first example

This section explains how to run a first example with STM32CubeL1, using as illustration the generation of a simple LED toggle running on the NUCLEO-L152RE board:

1. Download the STM32CubeL1 MCU Package. Unzip the package into a directory. Make sure not to modify the package structure shown in Figure 4
2. Browse to \Projects\NUCLEO-L152RE\Examples
3. Open the \GPIO folder, then open the \GPIO_EXTI folder
4. Open the project by selecting preferred toolchain
5. Rebuild all files and load the generated image into the target memory
6. Run the example: each time the Key push button is pressed, the LED2 toggles (for more details, refer to the example readme file).

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

- EWARM
  1. Under the example folder, open the \EWARM sub folder
  2. Open the Project.eww workspace (1)
  3. Rebuild all files: **Project**->**Rebuild all**
  4. Load the project image: **Project**->**Debug**
  5. Run the program: **Debug**->**Go** (F5)
- MDK-ARM
  1. Under the example folder, open the \MDK-ARM sub folder
  2. Open the Project.uvproj workspace(1)
  3. Rebuild all files: **Project**->**Rebuild all target files**
  4. Load the project image: **Debug**->**Start/Stop Debug Session**
  5. Run the program: **Debug**->**Run** (F5)
- SW4STM32
  1. Open the \SW4STM32 toolchain
  2. Click on **File->Switch Workspace->Other** and browse to the SW4STM32 workspace directory
  3. Click on **File->Import**, select **General->Existing Projects into Workspace** and then click **Next**
  4. Browse to the SW4STM32 workspace directory and select the **project**
  5. Rebuild all project files: select the project in the “**Project explorer”** window then click on **Project->build project** menu
  6. Run program: **Run->Debug** (F11)

1. *The workspace name may change from one example to another*

4.2 Developing your own application

4.2.1 HAL application

This section describes the steps required to create your own HAL application using STM32CubeL1:

1. **Create your project:** to create a new project, it is possible to 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, for example STM32L152D-EVAL).

   The Template project provides an empty main loop function. It is a good starting point to get familiar with the project settings for STM32CubeL1. The template has the following characteristics:

   a. It contains source code of the HAL, CMSIS and BSP drivers, that are the minimum components required to develop a code on a given board.

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

   c. It defines the STM32L1 device supported and allows configuring the CMSIS and HAL drivers accordingly.

   d. It provides ready-to-use user files, that are pre-configured as follows:

   - HAL is initialized
   - SysTick ISR implemented for HAL_Delay() purpose
   - System clock is configured with the minimum frequency of the device (MSI) and though for an optimum power consumption.

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 Device Libraries, STMTouch touch library, FreeRTOS™ and FatFS. To find out which source files are needed to add to the project files list, refer to the documentation provided for each Middleware. It is possible also to 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 has to be copied into 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 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 HAL_Init() API to initialize the HAL Library and do the following:

   a. Configure the Flash prefetch, instruction and data caches (user-configurable by macros defined in stm32l1xx_hal_conf.h)

   b. Configure the Systick to generate an interrupt every 1 msec, which is clocked by the MSI, this the default configuration after reset (at this stage, the clock is not yet configured and thus the system is running from the internal 4 MHz MSI).

   c. Call the HAL_MspInit() callback function defined in the user file stm32l1xx_hal_msp.c to do the global low-level hardware initialization

5. **Configure the system clock:** the system clock configuration is set by calling the two following 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. The user can also skip the PLL configuration if there is no need to run the system at high frequency

   b. HAL_RCC_ClockConfig(): configures the system clock source, Flash latency and AHB and APB prescalers.

Note: Prior to configuring the system clock, it is recommended to enable the power controller clock, and to configure the appropriate voltage scaling, and therefore to optimize the power consumption when the system is clocked below the maximum allowed frequency.

6. **Initialize the peripheral**

   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 stm32xxx_it.c to call required interrupt handlers (peripheral and DMA), if needed

c. Write process complete callback functions if peripheral interrupt or DMA has to be used

d. In the main.c file, initialize the peripheral handle structure then call the function HAL_PPP_Init() to initialize the peripheral

7. **Develop custom application:**At this stage, the system is ready and development of an application code can started.

a. The HAL provides intuitive and ready-to-use APIs to configure 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 extensive set of examples provided.

b. If the application has some real-time constraints, a large set of examples are available showing how to use FreeRTOS™ and how integrate it with all Middleware stacks provided in STM32CubeL1. This can be a good starting point for development.

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

4.2.2 LL application

This section describes the steps needed to create your own LL application using STM32CubeL1.

1. **Create your project**

To create a new project, it is possible to 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> refers to the board name, such as STM32L152D-EVAL).

The template project provides an empty main loop function, however it is a good starting point to get familiar with project settings for STM32CubeL1.

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 STM32L1 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 minimum frequency of the device (HSI) for an optimum power consumption.

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 1. Macros for STM32L1 devices, or to the application note “STM32Cube firmware examples for STM32L1 Series” (AN4706).

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 stm32l1xx_it.h file
- Replace stm32l1xx_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

An alternative to steps 1 to 6 described in Section 4.2 Developing your own application consists in using the STM32CubeMX tool to generate code for the initialization of the system, the peripherals and middleware (Steps 1 to 5 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 thanks to a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and the utility performing MCU peripheral configuration (for example GPIO, USART) and middleware stacks (for example USB).
3. Generate the initialization C code based on the configuration selected. This code is ready to use within several development environments. The user code is kept at the next code generation.

For more information, refer to STM32CubeMX user manual (UM1718).

4.4 Getting STM32CubeL1 release updates

The STM32CubeL1 MCU 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 www.st.com and proposes to download them to the user's computer.

4.4.1 Installing and running the STM32CubeUpdater program

- Double-click on the 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 Updated configuration, refer to section 3 of STM32CubeMX User manual (UM1718).

5 FAQs

5.1 What is the license scheme for the STM32CubeL1 MCU Package?

The HAL is distributed under a non-restrictive BSD (Berkeley software distribution) license.

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

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

5.2 What boards are supported by the STM32CubeL1 MCU Package?

The STM32CubeL1 MCU Package provides BSP drivers and ready-to-use examples for the STM32L152D-EVAL, 32L152CDISCOVERY, 32L100CDISCOVERY and NUCLEO-L152RE boards.

For the up-to-date list of supported boards, refer to the firmware package release notes.

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

Yes. STM32CubeL1 provides a rich set of examples and applications. They come with the pre-configured projects of several toolsets: IAR™, Keil® and GCC.

5.4 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 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 registers 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 allows an easier migration from the SPL to the STM32Cube LL drivers, since each SPL API has its equivalent LL API(s).

5.5 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).

5.6 How are the product/peripheral specific features managed?

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

5.7 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.

5.8 How to get regular updates on the latest STM32CubeL1 firmware releases?

The STM32CubeL1 MCU Package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new MCU Package updates (new releases or/and patches).

STM32CubeUpdater is integrated as well within the STM32CubeMX tool. When using this tool for STM32L1 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeL1 MCU Package updates.

For more details, refer to Section 4.4 Getting STM32CubeL1 release updates.

5.9 When should I use HAL versus LL drivers?

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

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

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

There is no configuration file. Source code shall directly include the necessary stm32l1xx_ll_ppp.h file(s).

5.11 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 to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

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

Yes, there are.

A few Cortex® APIs have been added in stm32l1xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

5.13 Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, SysTick interrupts has not to be enabled because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

5.14 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.

Revision history

Table 4. Document revision history

Date	Revision	Changes
02-Sep-2014	1	Initial release.
04-Feb-2015	2	Added STM32L151xDX, STM32L152xDX and STM32L162xDX macros in Table 1. Macros for STM32L1 devices.
11-Jun-2015	3	Added SW4STM32 in Section 3.2 MCU Package overview and Section 4.1 Running your first example.
01-Jun-2016	4	Updated Section Introduction, Section 3.2 MCU Package overview, Section 3.2 MCU Package overview and Section 5 FAQs. Updated Figure 1. STM32CubeL1 firmware components, Figure 2. STM32CubeL1 firmware architecture, Figure 3. STM32CubeL1 MCU Package structure and Figure 4. Projects for NUCLEO-L152RE board Updated Table 3. Number of examples for each board.
10-Apr-2017	5	Updated Section 3.2 MCU Package overview and Section 4.2 Developing your own application with introduction of Section 4.2.1 HAL application and Section 4.2.2 LL application. Updating Figure 1. STM32CubeL1 firmware components and Figure 4. Projects for NUCLEO-L152RE board. Updated Table 3. Number of examples for each board
08-Apr-2019	6	Updated: • Section Introduction • Table 1. Macros for STM32L1 devices • Figure 3. STM32CubeL1 MCU Package structure • Figure 4. Projects for NUCLEO-L152RE board • Section 4.1 Running your first example Removed reference to TrueSTUDIO in the whole document.

Contents

List of tables

List of figures