UM1802
User manual

Getting started with the STM32CubeL1 firmware package for the STM32L1 series

Introduction

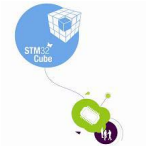
STMCube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time, cost. STM32Cube covers STM32 portfolio.

STM32Cube™ Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL1 for STM32L1 series)
  - The STM32CubeL1 HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, STMTouch, FatFS
  - All embedded software utilities coming with a full set of examples.

This user manual describes how to get started with the STM32CubeL1 firmware package:

- *Section 1* describes the main features of STM32CubeL1 firmware, part of the STM32Cube initiative.
- *Section 2* and *Section 3* provide an overview of the STM32CubeL1 architecture, firmware package structure.



Contents

List of tables

List of figures

1 STM32CubeL1 main features

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 series 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 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 an open-source BSD license for user convenience.

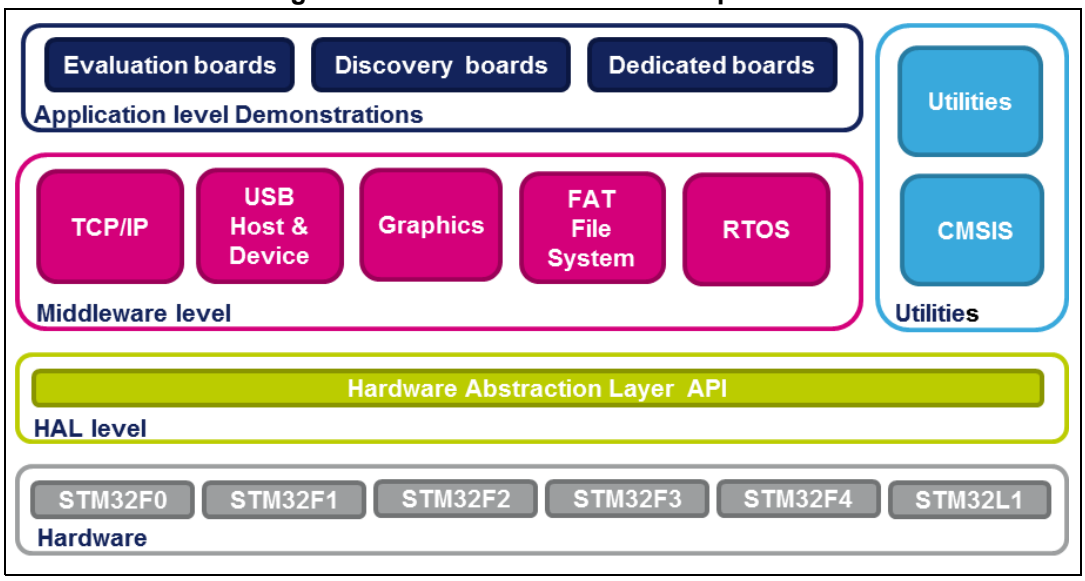
STM32CubeL1 package features a set of middleware components with the corresponding examples. They come with very permissive license terms:

- Full USB Device stack supporting many classes (HID, MSC, CDC, DFU);
- CMSIS-RTOS implementation with FreeRTOS open source solution;
- FAT File system based on open source FatFS solution;
- STMTouch touch sensing solution
- STemWin, a professional graphical stack solution available in binary format, based on ST partner solution SEGGER emWin.

Several applications, demonstrations implementing all these middleware components are also provided in the STM32CubeL1 package.

The block diagram of STM32Cube is shown in *Figure 1*.

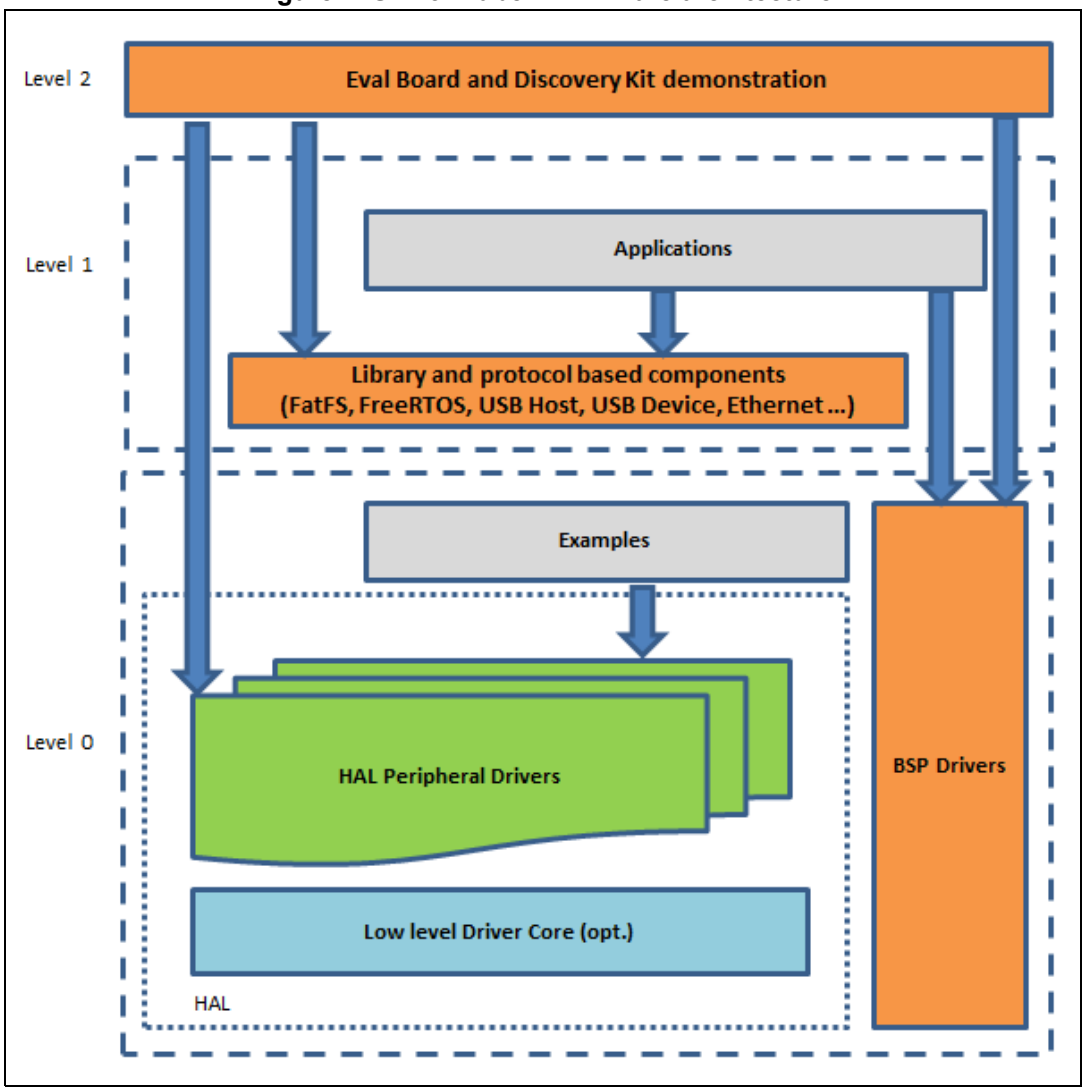
Figure 1. STM32Cube firmware components



2 STM32CubeL1 architecture overview

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

Figure 2. STM32CubeL1 firmware architecture



Level 0 is divided in three sub-layers:

- **Board Support Package (BSP)**: this layer offers a set of APIs relative to the hardware components in the hardware boards (LCD drivers, microSD, etc…), composed of two parts:
  - Component: is the driver relative to the external device on the board, not related to the STM32, the component driver provide specific APIs to the BSP driver external components, could be portable on any other board.
  - BSP driver: it allows to link the component driver to a specific board, provides a set of friendly used APIs. The APIs naming rule is BSP_FUNCT_Action(), so we have, as examples, BSP_LED_Init(), BSP_LED_On();

  BSP layer is based on modular architecture allowing an easy porting on any hardware by just implementing the low level routines.
- **Hardware Abstraction Layer (HAL)**: this layer provides the low level drivers, the hardware interfacing methods to interact with the upper layers (application, libraries, stacks). It provides generic, multi instance, functionalities oriented APIs which permit to offload the user application implementation by providing ready to use process. As example, for the communication peripherals (I2S, UART…) it provides APIs allowing to initialize, configure the peripheral, manage data transfer based on polling, interrupt or DMA process,, manage communication errors that may raise during communication. The HAL Drivers APIs are split in two categories, generic APIs which provides common, generic functions to all the STM32 series, extension APIs which provides specific, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer encloses the examples build over the STM32 peripheral using only the HAL, BSP resources.

Level 1 is divided in two sub-layers:

- **Middleware components**: set of Libraries covering USB Device library, STMTouch touch sensing library, graphical STemWin library, FreeRTOS, FatFS.

  Horizontal interaction 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, static macros implemented in the library system call interface.

  As 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 the following:

  USB Device Library

  – Supports several USB classes (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, the library configuration without changing the library code (Read Only).
  – RTOS, 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.

  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, standalone operation;
  – Examples with microSD.

  STM32 Touch Sensing Library

  – Robust STMTouch capacitive touch sensing solution supporting proximity, touchkey, linear, rotary touch sensor using a proven surface charge transfer acquisition principle.

  STemWin Library

  – Graphical library supporting LCD provided as part as the STM32CubeL1 firmware package.

- 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: is composed of a single layer which is a global real-time, graphical demonstration based on the Middleware service layer, the low level abstraction layer, the basic peripheral usage applications for board based functionalities.

3 STM32CubeL1 firmware package overview

3.1 Supported STM32L1 devices, hardware

STM32Cube™ is offering highly portable Hardware Abstraction Layer (HAL) built around a generic architecture, allows the build-upon layers, like the middleware layer, to implement its functions without knowing, in-depth, the MCU used. This improves the library code reusability, guarantees an easy portability on other devices.

Thanks to its layered architecture, the STM32CubeL1 offers full support to all microcontrollers of the STM32L1 series, user only needs to define the right macro in stm32l1xx.h.

Table 1 below provides the macro to be defined depending on the used microcontroller.

Table 1. Macros for STM32L1 series

Macro 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
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
STM32L162xC	STM32L162RC, STM32L162VC
STM32L162xCA	STM32L162RC-A, STM32L162VC-A, STM32L162QC, STM32L162ZC
STM32L162xD	STM32L162QD, STM32L162RD, STM32L162VD, STM32L162ZD
STM32L162xE	STM32L162RE, STM32L162VE, STM32L162ZE

STM32CubeL1 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 boards listed in *Table 2*.

Table 2. Boards for STM32L1 series

Board	Supported devices
STM32L152D-EVAL	STM32L152xD
STM32L152C-Discovery	STM32L152xC
STM32L100-Discovery	STM32L100xC
NUCLEO-L152RE	STM32L152xE

As for all other STM32 Nucleo boards, the NUCLEO-L152RE features a reduced set of hardware components (one user Key button and one user LED). In order to enrich the middleware support offer for in STM32CubeL1 firmware package, an LCD display Adafruit Arduino shield was chosen, which embeds a µSD connector and a Joystick in addition to the LCD.

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.

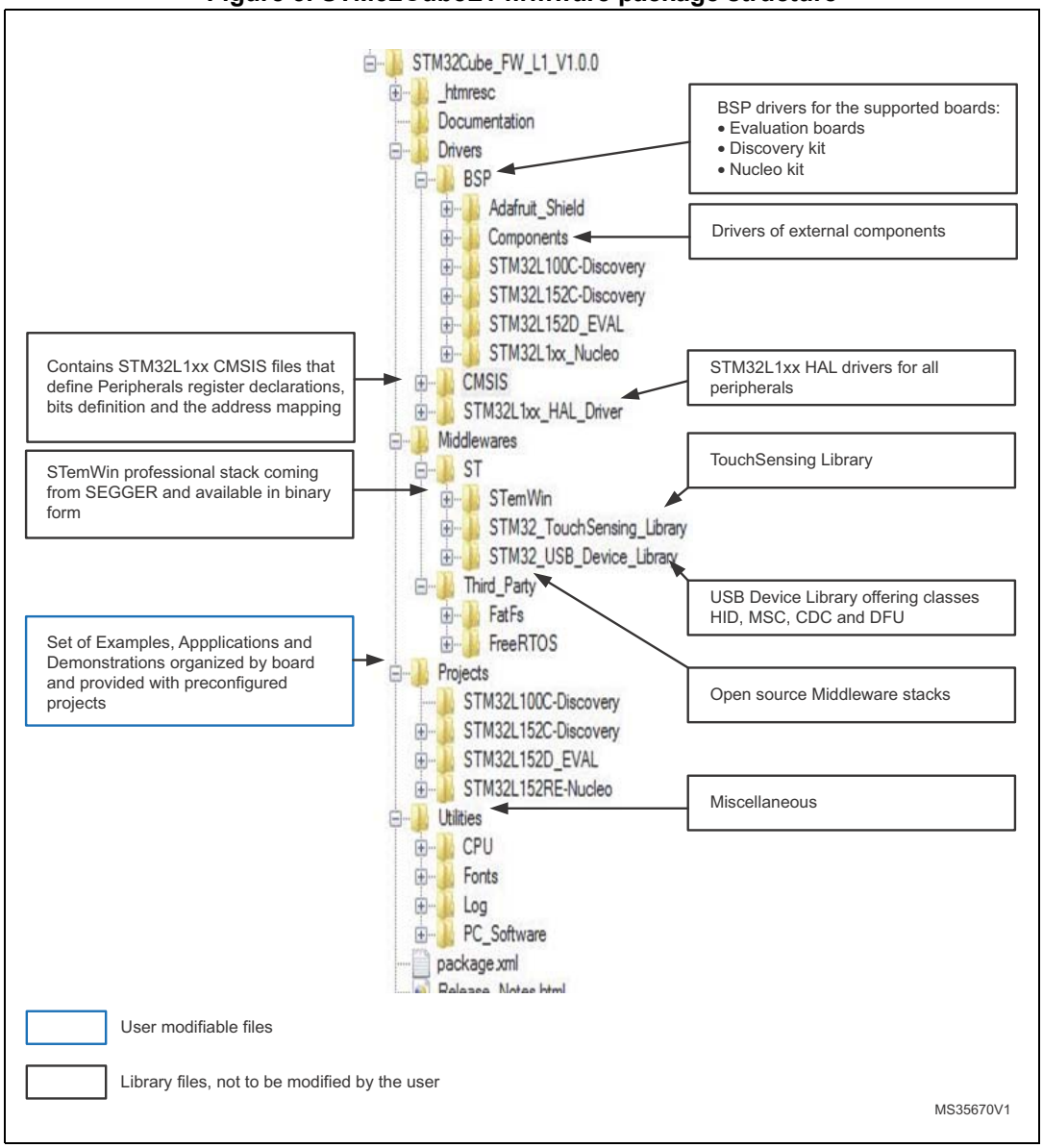
The STM32CubeL1 firmware is able to run on any compatible hardware. That means user can simply update the BSP drivers to port the provided examples on his own board, if this later has the same hardware functionalities (LED, LCD Display, Buttons...etc.).

3.2 Firmware package overview

The STM32CubeL1 firmware solution is provided in one single zip package having the structure shown in *Figure 3*.

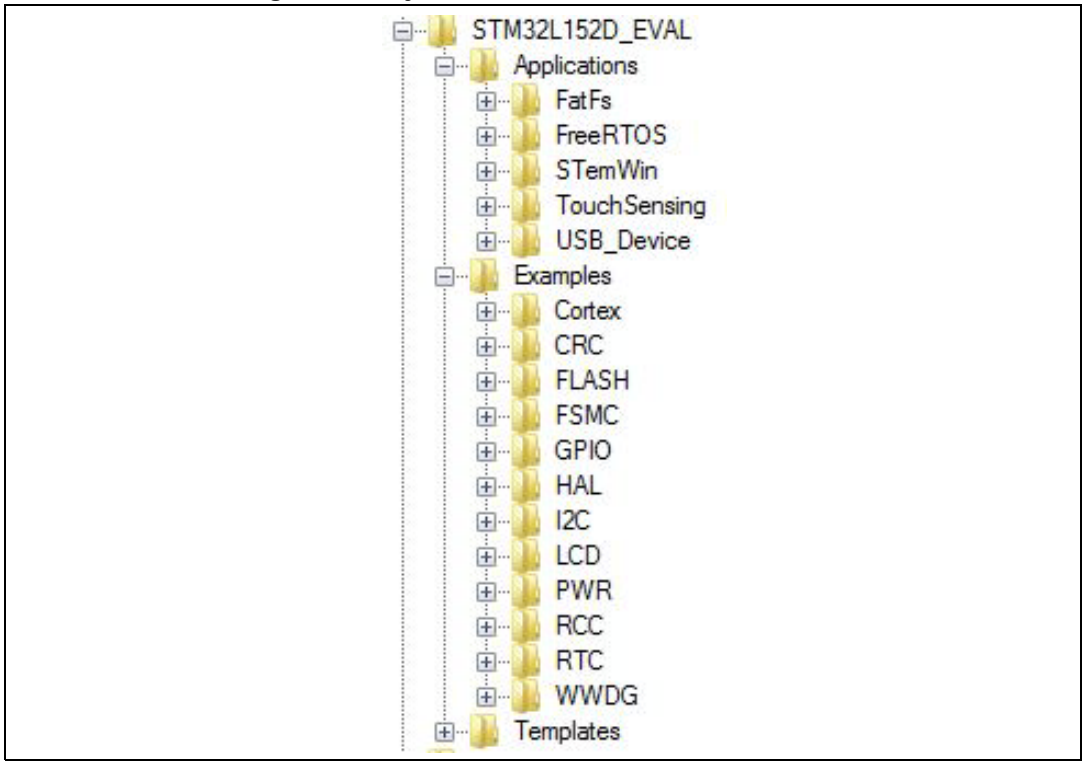
.

Figure 3. STM32CubeL1 firmware package structure



For each board, a set of examples are provided with preconfigured projects for EWARM, MDK-ARM™ and TrueSTUDIO® toolchains, as exemplified in *Figure 4*.

Figure 4. Projects for STM32L152D-EVAL board



The examples are classified depending on the STM32Cube level they apply to:

- those in level 0 are called **Examples**, they use HAL drivers without any Middleware component;
- the ones in level 1 are called **Applications**, they provide typical use cases for each Middleware component.

The **Template** project available in the Template directory is provided 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 **\TrueSTUDIO** folders contain the preconfigured project for each toolchain;
- **readme.txt**, describing the behavior and needed environment to make them working.

Table 3 provides a numerical summary of available examples.

Table 3. Number of examples for each board

Board	Examples	Applications	Demonstration
STM32L152D-EVAL	29	17	N/A
STM32L152C-Discovery	8	1	1
STM32L100-Discovery	7	1	1
NUCLEO-L152RE	22	3	1

4 Getting started

4.1 Running your first example

This section shows how simple is to run a first example within STM32CubeL1, using as an example the generation of a simple LED toggle running on STM32L152RE Nucleo board:

1. Download the STM32CubeL1 firmware package. Unzip it into a directory of your choice. Make sure not to modify the package structure shown in Figure 4 above). Note that it is also recommended to copy the package at a location close to your root volume (i.e. C\Eval or G:\Tests) because some IDEs encounter problems when path length is too high.
2. Browse to \Projects\STM32L152RE-Nucleo\Examples
3. Open \GPIO, then \GPIO_EXTI folder
4. Open the project with your preferred toolchain (*)
5. Rebuild all files and load your image into target memory
6. Run the example: each time you press the USER push button, the LED2 toggles (for more details, refer to the example readme file).

(*) Below 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 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 project image: **Debug->Start/Stop Debug Session**
  - Run program: **Debug->Run (F5)**
- TrueSTUDIO
  - Open the TrueSTUDIO toolchain
  - Click on **File->Switch Workspace->Other** and browse to TrueSTUDIO workspace directory
  - Click on **File->Import**, select **General->'Existing Projects into Workspace'** and then click **Next**.
  - Browse to the TrueSTUDIO 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 change from one example to another.

4.2 How to develop your own application

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

1. **Create your project**: for a new project you can either start from the Template project provided for each board under \Projects\<STM32xxx_yyy>\Templates or from any available project under \Projects\<STM32xxy_yyy>\Examples or \Projects\<STM32xx_yyy>\Applications (<STM32xxx_yyy> refers to the board name, such as STM32L152D_EVAL).

   The Template project is providing empty main loop function, however it's a good starting point to get familiar with project settings for STM32CubeL1. The main characteristics are listed below:

   a) It contains sources of HAL, CMSIS and BSP drivers which are the minimal components 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, allowing to configure the CMSIS and HAL drivers accordingly

   d) It provides ready to use user files preconfigured as follows:

   – HAL initialized with default time base with ARM® Core SysTick;

   – SysTick ISR implemented for HAL_Delay() purpose;

   – System clock configured with the minimum frequency of the device (HSI) for an optimum power consumption.

Note: If you copy an existing project to another location, then you need to update the include paths.

2. **Add the necessary Middleware to your project (optional)**: available Middleware stacks are:

   – USB Device Library

   – STemWin

   – Touch Sensing Library

   – FreeRTOS

   – FatFS.

   To know which source files you need to add in the project files list; refer to the documentation provided for each Middleware, you may have a look also to the Applications available under \Projects\STM32xxx_yyy\Applications\<MW_Stack> (<MW_Stack> refers to the Middleware stack, ex USB_Device) to have better idea on the sources 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 "#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, the word "_template" need to be removed when copying it to the project folder).

   The configuration file provides enough information to evaluate 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 need to call HAL_Init() API to initialize the HAL Library, which do the following:
   a) Configure the Flash prefetch and SysTick interrupt priority (configured by user through macros defined in stm32l1xx_hal_conf.h)
   b) Configure the Systick to generate an interrupt each 1 msec at the SysTick interrupt priority TICK_INT_PRIO defined in stm32L1xx_hal_conf.h, 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 8 MHz)
   c) Set NVIC Group Priority to 4
   d) Calls HAL_MspInit() callback function defined in user file stm32l1xx_hal_msp.c to do global low level hardware initializations.
5. **Configure the system clock**: this is done by calling these two APIs:
   a) HAL_RCC_OscConfig(): configures the internal and/or external oscillators, PLL source and factors. User may select to configure one oscillator or all oscillators, in addition PLL configuration may be skipped 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 prescaler.
6. **Peripheral initialization**:
   a) Start by writing the peripheral HAL_PPP_MspInit function. For this function, please 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 you plan to use peripheral interrupt or DMA.
   d) In your main.c file, initialize the peripheral handle structure then call the function HAL_PPP_Init() to initialize your peripheral.
7. **Developing your application process**: at this stage, your system is ready and you can start developing your application code.
   a) The HAL provides intuitive and ready to use APIs to configure the peripheral, and supports polling, IT and DMA programming model, to accommodate any application requirements. For more details on how to use each peripheral, refer to the rich examples set provided.
   b) If your application has some real time constraints, you can found a large set of examples showing how to use FreeRTOS and its integration with all Middleware stacks provided within STM32CubeL1, it can be a good starting point for your 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 please refer to HAL_TimeBase example.

4.3 Using STM32CubeMX to generate the initialization C code

An alternative to steps *1.* to *6.* described in *Section 4.2* consists in using the STM32CubeMX tool to generate code for the initialization of the system, the peripherals and middleware through a step-by-step process:

- select the STM32 microcontroller that matches the required set of peripherals;
- 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);
- 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 informations, please refer to UM1718.

4.4 Getting STM32CubeL1 release updates

The STM32CubeL1 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 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 a 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 FAQ

What is the license scheme for the STM32CubeL1 firmware?

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license. The Middleware stacks made by ST (USB Device, Touch Sensing and STemWin libraries) 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.

What boards are supported by the STM32CubeL1 firmware package?

The STM32CubeL1 firmware package provides BSP drivers and ready-to-use examples for the following STM32L1 boards: STM32L152D-EVAL, STM32L152C-Discovery, STM32L100-Discovery and NUCLEO-L152RE.

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.

Customers currently using Standard Peripheral Libraries will be helped through Migration guides. 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. 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. STM32CubeL1 provides a rich set of examples and applications (around 90 in total). They come with the pre-configured project of several toolsets, namely IAR™, Keil® and GCC.

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.

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 STM32CubeL1 firmware releases?

The STM32CubeL1 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 STM32L1 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeL1 firmware package updates.

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

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
02-Sep-2014	1	Initial release.