UM1754
User manual

Getting started with STM32CubeL0 firmware package for STM32L0xx series

Introduction

The STM32Cube™ initiative was originated by STMicroelectronics to ease developers life by reducing development efforts, time and cost. STM32Cube™ covers the 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 STM32CubeL0 for STM32L0 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across the STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, STMTouch and FatFS
  - All embedded software utilities coming with a full set of examples.

This user manual describes how to get started with the STM32CubeL0 firmware package.

Chapter 1 describes the main features of STM32CubeL0 firmware, part of the STM32Cube™ initiative.

Chapter 2 and *Chapter 3* provide an overview of the STM32CubeL0 architecture and firmware package structure.

Contents

List of tables

List of figures

1 STM32CubeL0 main features

STM32CubeL0 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32L0 microcontrollers. In line with the STM32Cube initiative, this set of components is highly portable, not only within the STM32L0 series but also to other STM32 series.

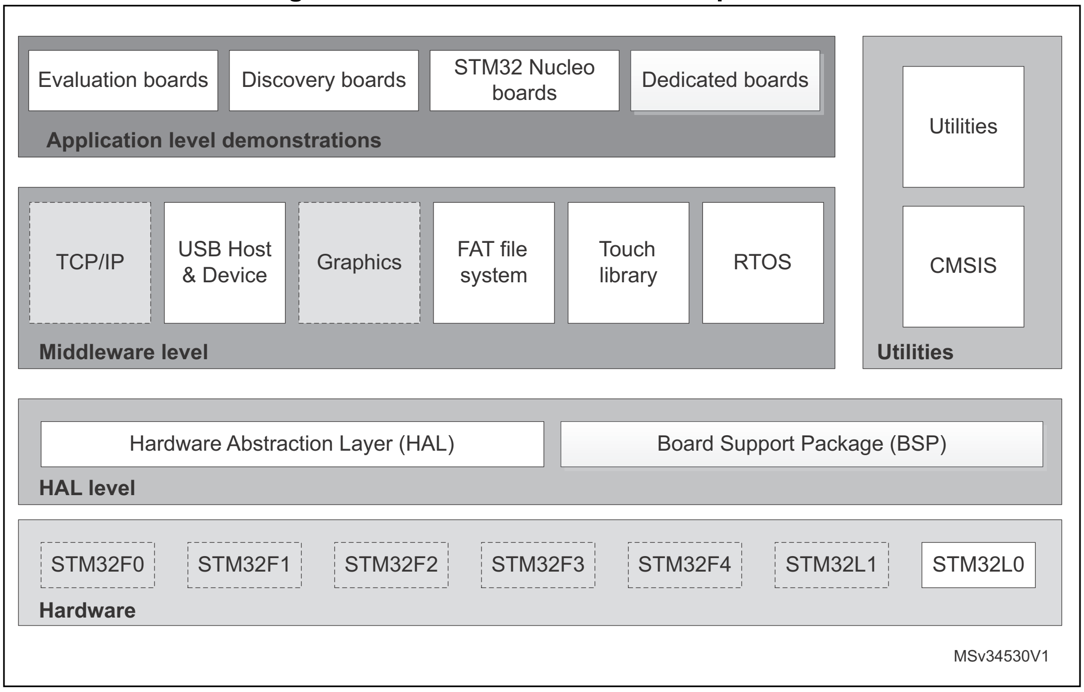
STM32CubeL0 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in an open-source BSD license for user convenience.

STM32CubeL0 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: HID, MSC, CDC, Audio, DFU
- CMSIS-RTOS implementation with FreeRTOS open source solution
- FAT File system based on open source FatFS solution
- STMTouch touch sensing solutions

A demonstration implementing all these middleware components is also provided in the STM32CubeL0 package.

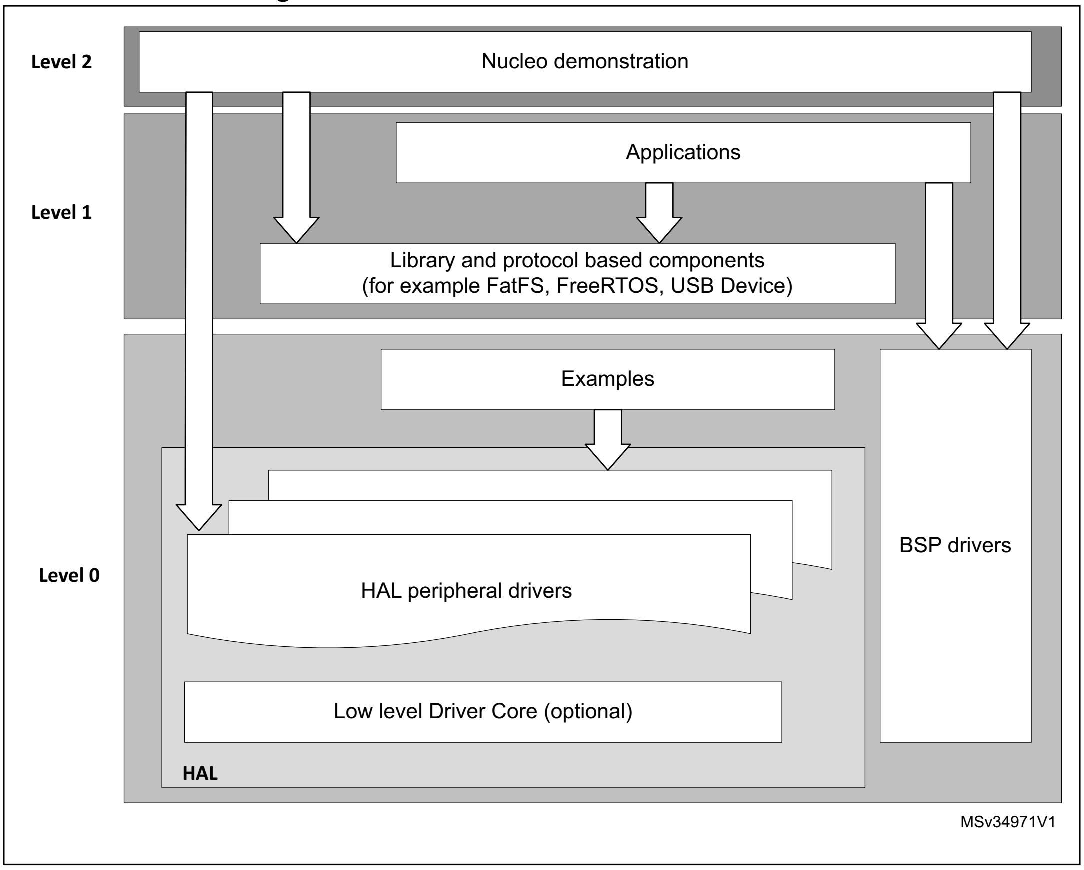
Figure 1. STM32Cube firmware components



2 STM32CubeL0 architecture overview

The STM32CubeL0 firmware solution is built around three independent levels that can easily interact with each other as described in the *Figure 2* below:

Figure 2. STM32CubeL0 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 (LCD drivers, Micro SD. etc…) and composed of two parts:
  - Component: is the driver related to the external device on the board and not related to the STM32, the component driver provides specific APIs to the BSP driver's external components and can be ported to any board.
  - BSP driver: it enables the component driver to be linked to a specific board and provides a set of user-friendly APIs. The API naming rule is BSP_FUNCT_Action(): ex. BSP_LED_Init(),BSP_LED_On()

  It's based on a modular architecture that allows it to be ported easily to any hardware by just implementing the low level routines.
- **Hardware Abstraction Layer (HAL):** this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi instance and function-oriented APIs which allow to offload the user application implementation by providing ready-to-use processes. For

example, for the communication peripherals (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 in two categories, generic APIs which provides common and generic functions to all the STM32 series and extension APIs which provides 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 STM32L0 peripherals using only the HAL and BSP resources.

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

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

  The main features of each Middleware component are as follow:

  USB Device Library
  - Several USB classes supported (Mass-Storage, HID, CDC, DFU, AUDIO, MTP)
  - Supports multi packet transfer features: allows sending big amounts of data without splitting them into max packet size transfers.
  - Uses configuration files to change the core and the library configuration without changing the library code (Read Only).
  - RTOS and Standalone operation
  - The link with low-level driver is done through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level drivers.

  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 sensor using a proven surface charge transfer acquisition principle.

- **Examples based on the Middleware components:** each Middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several Middleware components are provided as well.

Level 2: This level is composed of a single layer which is a global real-time and graphical demonstration based on the Middleware service layer, the low level abstraction layer and the applications that make basic use of the peripherals for board-based functions.

3 STM32CubeL0 firmware package overview

3.1 Supported STM32L0 devices and hardware

STM32Cube offers a highly portable Hardware Abstraction Layer (HAL) built around a generic and modular architecture allowing the upper layers, Middleware and Application, to implement their functions without in-depth knowledge of the MCU being used. This improves the library code re-usability and guarantees an easy portability from one device to another.

The STM32CubeL0 offers full support for all STM32L0 Series devices. The user only has to define the right macro in stm32l0xx.h.

Table 1 below lists the macro to define depending on the STM32L0 device in use. Note that the macro must also be defined in the compiler preprocessor.

Table 1. Macros for STM32L0 series

Macro defined in stm32l0xx.h	STM32L0 devices
STM32L051xx	STM32L051K8, STM32L051C6, STM32L051C8, STM32L051R6 and STM32L051R8
STM32L052xx	STM32L052K6, STM32L052K8, STM32L052C6, STM32L052C8, STM32L052R6 and STM32L052R8
STM32L053xx	STM32L053C6, STM32L053C8, STM32L053R6 and STM32L053R8
STM32L062xx	STM32L062K8
STM32L063xx	STM32L063C8 and STM32L063R8

STM32CubeL0 features a rich set of examples and applications making it easy to understand and use any HAL driver and/or Middleware components. These examples can be run on any of the STMicroelectronics board as listed in *Table 2* below:

Table 2. Nucleo board for STM32L0 series

Board	STM32L0 devices supported
NUCLEO-L053R8	STM32L053x8

As for all other STM32 Nucleo boards, the NUCLEO-L053R8 features a reduced set of hardware components (one user Key button and one user LED). And in order to enrich the middleware support offer for in STM32CubeL0 firmware package, an LCD display Adafruit Arduino shield was chosen, which embeds in addition to the LCD a µSD connector and Joystick.

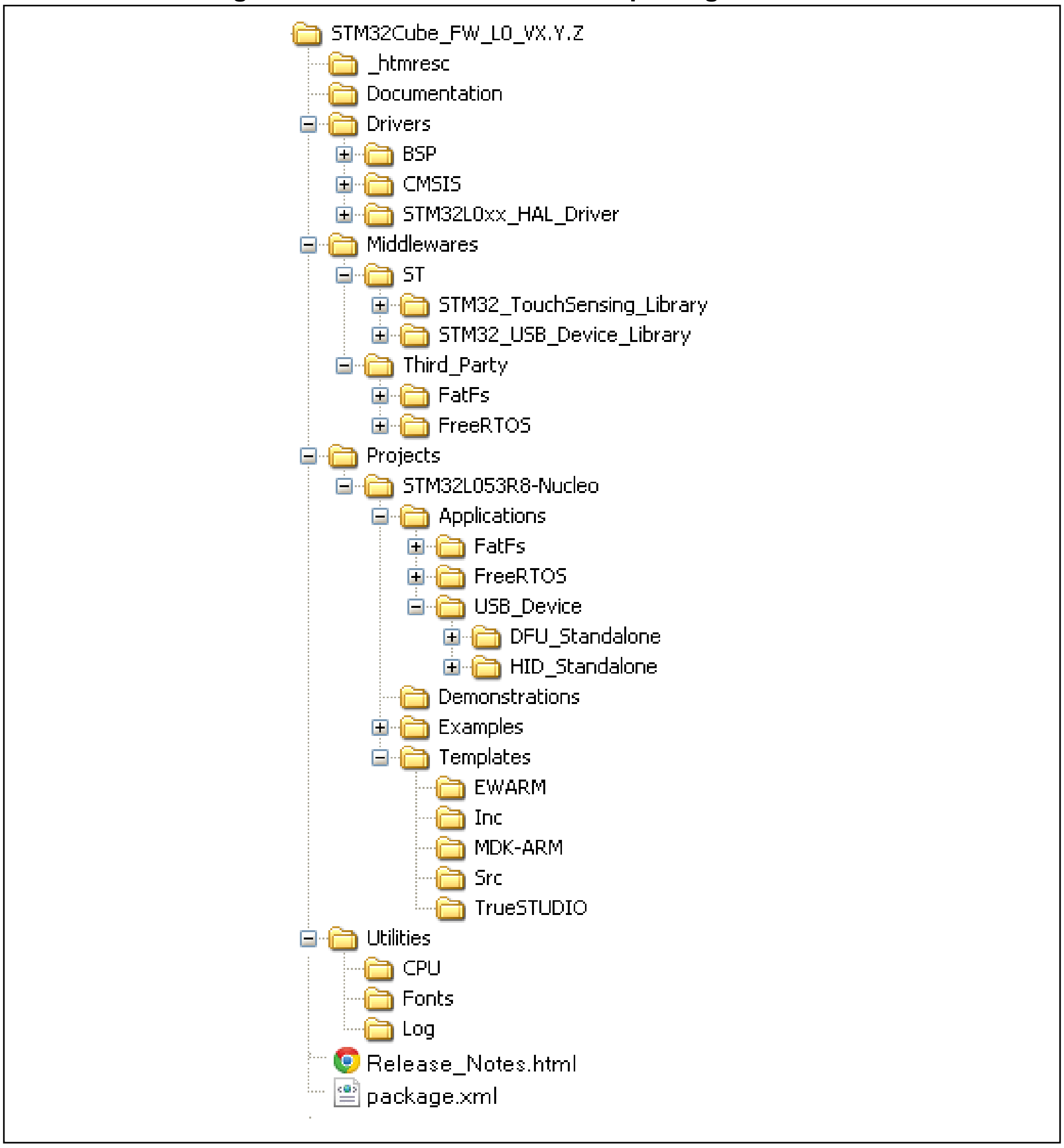
In the BSP component the dedicated drivers, for that Arduino shield are available and their use is illustrated through either the provided BSP example or in the Demonstration firmware, without forgetting the FatFS middleware application.

The STM32CubeL0 firmware is able to run on any compatible hardware. The users can simply update the BSP drivers to port the provided examples to their own board, providing it has the same hardware functions (for example LED, pushbuttons).

3.2 Firmware package overview

The STM32CubeL0 firmware solution is provided in a single zip package with the structure shown in *Figure 3* below.

Figure 3. STM32CubeL0 firmware package structure



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

Figure 4 shows the project structure for the NUCLEO-L053R8 board. The structure is identical for any other additional supported 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, that use HAL 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

A template project is provided to users for the quick build of 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 containing the preconfigured project for each toolchain.
- *readme.txt* describing the example behavior and the required environment to make it work

Figure 4. STM32CubeL0 examples overview

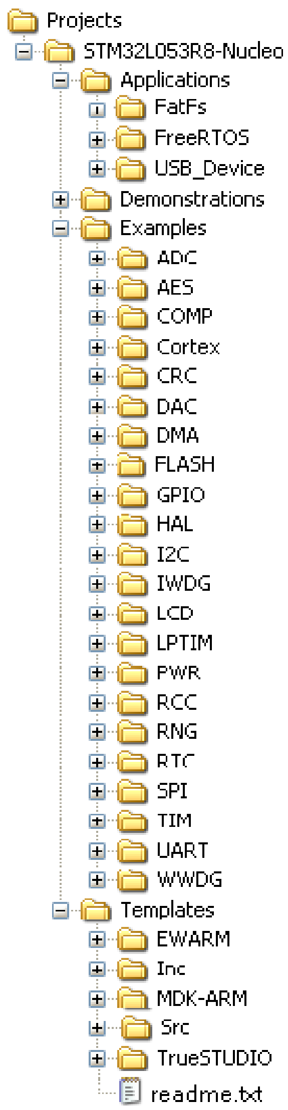


Table 3 provides the number of examples, applications and demonstrations available for the NUCLEO-L053R8 board.

All the examples are developed and optimized to consider power consumption constraints, which explains the choice a default system clock configuration to MSI, unless higher frequencies required by some given peripherals (for example USB, RNG).

Table 3. Number of examples available for each board

Board	Examples	Applications	Demonstration
NUCLEO-L053R8	69	10	1

4 Getting started

4.1 Running your first example

This section explains how to run a first example with STM32CubeL0., using as illustration the generation of a simple LED toggle running on the STM32L053R8 Nucleo board:

1. Download the STM32CubeL0 firmware package. Unzip the package into a directory of your choice. Make sure not to modify the package structure shown in *Figure 3*.
2. Browse to `\Projects\STM32L053R8-Nucleo\Examples`.
3. Open the *\GPIO* folder*,* then open the *\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 Key push button, 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
  - Under the *example* folder, open the *\EWARM* sub folder
  - Open the *Project.eww* workspace[(a)]
  - Rebuild all files: **Project**->**Rebuild al**l
  - Load the project image: **Project**->**Debug**
  - Run the program: **Debug**->**Go**(F5)
- MDK-ARM
  - Under the example folder, open the *\MDK-ARM* sub folder
  - Open the *Project.uvproj* workspace[(a)]
  - Rebuild all files: **Project**->**Rebuild all target files**
  - Load the project image: **Debug**->**Start/Stop Debug Session**
  - Run the program: **Debug**->**Run** (F5)
- TrueSTUDIO
  - Open the TrueSTUDIO toolchain
  - Select on **File**->**Switch Workspace**->**Other** and browse to the 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 the program: **Run**->**Debug** (F11)

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

4.2 Developing your own application

This section describes the successive steps to create your own application using STM32CubeL0.

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 (<STM32xx_xxx> refers to the board name, for example STM32L053R8).

   The Template project provides an empty main loop function. It is a good starting point to get familiar with the project settings for STM32CubeL0. It has the following characteristics:

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

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

   c) It defines the STM32L0 device supported, allowing to set the configuration for the CMSIS and HAL drivers respectively.

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

   - HAL is initialized

   - SysTick ISR implemented for HAL_Delay() purpose

   - System clock is configured with the minimum frequency of the device (MSI) and though for an optimum power consumption.

Note: If you copy an existing project to another location make sure to update the include paths.

2. **Adding the necessary Middleware to your project (optional):** the available Middleware stacks are: USB Device Libraries, STMTouch touch library, FreeRTOS and FatFS. To find out which source files you need to add to the project files list, refer to the documentation provided for each Middleware. You can also 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. **Configuring 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 you have to copy 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. **Starting 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 *stm32l0xx_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
   c) Call the *HAL_MspInit()* callback function defined in the user file *stm32l0xx_hal_msp.c* to do the global low level hardware initialization
5. **Configuring 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. **Peripheral initialization**
   a) Start by writing the peripheral HAL_PPP_MspInit function. For this function, please proceed as follows:
      i. Enable the peripheral clock.
      ii. Configure the peripheral GPIOs.
      iii. Configure DMA channel and enable DMA interrupt (if needed).
      iv. 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, 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 your application has some real-time constraints, you can find a large set of examples showing how to use FreeRTOS and integrate it with all Middleware stacks provided in STM32CubeL0. This 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 (steps 1 to 6 above) through a step-by-step process:

- Select the STMicroelectronics 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 information, please refer to UM1718.

4.4 Getting STM32CubeL0 release updates

The STM32CubeL0 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 an STMicroelectronics program under Program Files and is automatically launched.

The STM32CubeUpdater icon appears in the system tray:

- Right-click the updater icon and select **Updater Settings** to configure the Updater connection and whether to perform manual or automatic checks (see STM32CubeMX User guide - UM1718 section 3 - for more details on Updater configuration).

5 FAQs

What is the license scheme for the STM32CubeL0 firmware?

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.

What boards are supported by the STM32CubeL0 firmware package?

The STM32CubeL0 firmware package provides BSP drivers and ready-to-use examples for the NUCLEO-L053R8 board.

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

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. STM32CubeL0 provides a rich set of examples and applications (80 for NUCLEO-L053R8). 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, 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 STM32CubeL0 firmware releases?

The STM32CubeL0 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 STM32L0 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeL0 firmware package updates.

For more details, refer to *Section 4.4*.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
24-Apr-2014	1	Initial release.