# Getting started with the software package for the STEVAL-2STPD01 USB Type-C™ Power Delivery dual port adapter kit

## Introduction

The STSW-2STPD01 software package contains the application source code and libraries designed to demonstrate the capabilities of the STEVAL-2STPD01 USB Type-C™ Power Delivery dual port adapter kit.

The application firmware runs on the mainstream ARM® Cortex®-M0+ 32-bit STM32G071RBT6 microcontroller embedded in the NUCLEO-G071RB included in the STEVAL-2STPD01 kit. Thanks to the featured STM32CubeG0 USB PD 3.0 middleware stack, the kit is compliant with the USB Type-C 2.1 and Power Delivery 3.1 specifications.

The software package includes the firmware drivers of the STPD01 DC-DC converter and the TCPP02-M18 protection which manage two USB Type-C ports and are driven by the microcontroller through a set of dedicated APIs.

The STSW-2STPD01 embeds two software IPs (Power Sharing and Power Monitor modules) which allow the STM32G071RBT6 microcontroller to optimize the input power budget through the two ports. By using the STM32CubeMonUCPD GUI, the Power Sharing module allows setting the input power rating (as input voltage and current delivered by the power supply), enabling, for each port, the negotiable PDOs for the fixed rating. It permits the microcontroller to manage the two STPD01 DC-DC converters, associated with each USB Type-C port, and dynamically handle the available output according to the power negotiated in the explicit contract.

At the maximum input power rating (24 V, 6 A), the application firmware enables the adapter kit to deliver four fixed PDOs for each port: 5 V at 3 A, 9 V at 3 A, 15 V at 3 A, 20 V at 3 A.

# 1 Overview

The STSW-2STPD01 software package features:

- USB PD middleware stack based on STM32CubeG0 STM32Cube MCU Package for STM32G0 series running on the ARM® Cortex®-M0+ 32-bit STM32G071RBT6 microcontroller
- Software IP including the Power Monitor module
- Software IP including the Power Sharing module

**RELATED LINKS**

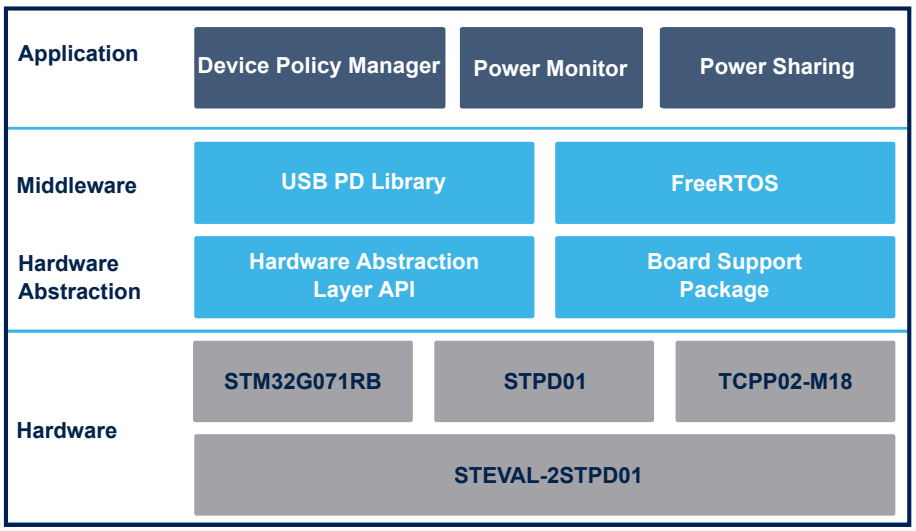
*UM2552:" Managing USB power delivery systems with STM32 microcontrollers"*

*Visit the wiki page for relevant guides and resources regarding USB Power Delivery*

# 2 Architecture

The STSW-2STPD01 architecture is organized in different levels, as shown in the following figure.

**Figure 1. STSW-2STPD01 software architecture**




1. **Hardware abstraction**
   - **STM32CubeG0 HAL** - hardware abstraction layer containing the device libraries specific for the STM32G0 microcontroller
   - **Board Support Package** - layer software and APIs to handle the STEVAL-2STPD01 and all the firmware drivers of the main integrated devices (STPD01 and TCPP02-M18)
2. **Middleware**
   - **USB PD Library** - divided in:
     - **Core** - provided as source code and compiled library and containing the ST USB PD middleware stack main blocks:
       - **DPM Core** - containing the main state machines and APIs to link the application layer
       - **Policy Engine** - to implement the local policy for a specific USB PD port
       - **Protocol Layer** - to enable messages to be exchanged between a Source port and a Sink port
     - **Device** - containing an adaptation layer for the STM32CubeG0 MCU:
       - **Physical Layer** - to handle transmission and reception of bits on the wire and data transmission
       - **USB-C Port Control** - to handle the Type-C detection state machines
   - **FreeRTOS** - containing several APIs to work with tasks, queues, semaphores, etc. as well as scheduler functions
3. **Application**
   - **Device Policy Manager** - to manage USB PD resources within the device across one or more ports based on the device local policy
   - **Power Monitor** - to monitor, at higher level, the bus status acquiring the voltage-current pair and reaching events (notifications and faults)
   - **Power Sharing** - specific IP to manage and distribute the power between the two ports, starting from the input power budget and considering the operative status of each port

The software architecture exploits the characteristics of FreeRTOS that embeds a scheduler and a task organization.

The application project, containing source files, linker configuration files, microcontroller startup file, etc., has been provided for the following IDEs: STM32CubeIDE, EWARM (IAR) and µVision (Keil).

## RELATED LINKS

*UM2552:" Managing USB power delivery systems with STM32 microcontrollers"*

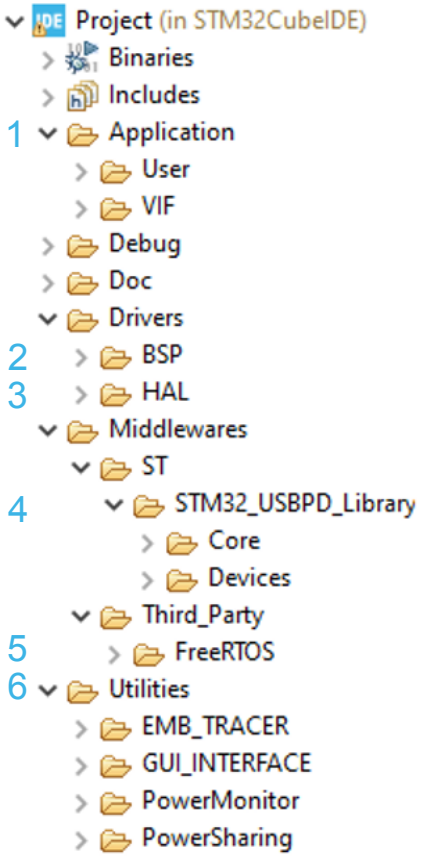
*4 Workspaces on page 12*

# 3 Project folder structure

The following figure shows the project main file organization and the related folder structure.

**Figure 2. Project folders and file organization**

1. Application user files
2. BSP and modules
3. HAL drivers
4. USB PD stack library
5. FreeRTOS source code
6. Utilities collection

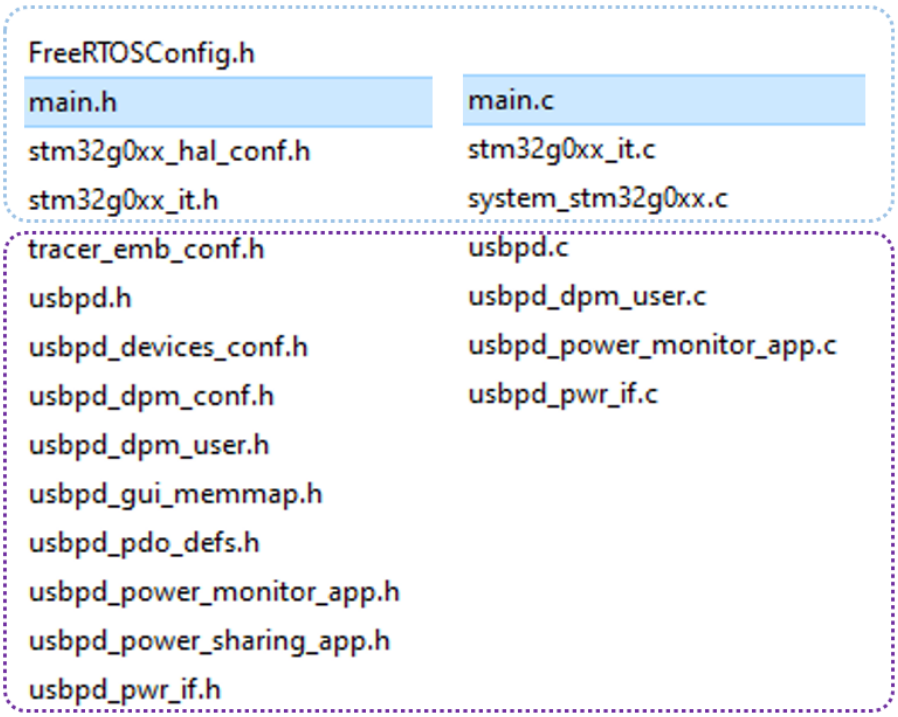



## 3.1 Application

The application user code is divided in:

- application files, containing main and system files (highlighted with dotted blue lines in the figure below)
- USB PD user files (highlighted with dotted purple lines in the figure below) dedicated to user settings and library configuration

Figure 3. **Application files**




### 3.1.1 Main and system files

Table 1. **Main and system files**

| File name | Description |
|---|---|
| *Main.c / .h* | Firmware application entry point that allows managing the hardware/firmware configuration and the boot sequence |
| *FreeRTOSConfig.h* | FreeRTOS configuration header file |
| *stm32g0xx_it.c / .h* | Interrupt routine service files |
| *system_stm32g0xx.c* | System package file (1) |
| *stm32g0xx_hal_conf.h* | HAL configuration header file |

1. *For further details refer to STM32Cube.*

### 3.1.2 USB PD user files

These user application files configure the USB PD library provided with the STM32Cube MCU Package for STM32G0 and implement the required callbacks/handlers.

Table 2. **USB PD user files**

| File name | Description |
|---|---|
| *usbpd.c / .h* | USB PD user file containing library configuration, GUI initialization and version |
| *usbpd_devices_conf.h* | USB PD user file containing the device defines |
| *usbpd_dpm_conf.h* | USB PD DPM configuration file enabling the Type-C port, VID, PID, etc. |
| *usbpd_dpm_user.c / .h* | DPM user implementation files containing all notification callbacks from and required by USB PD stack library core |
| *usbpd_gui_memmap.h* | GUI flash memory configuration |

| File name | Description |
|---|---|
| *usbpd_pdo_defs.h* | PDO definitions |
| *usbpd_pwr_if.c / .h* | Power interface files containing all APIs to turn $V_{BUS}$ on/off, to set a new profile and check the bus status |
| *usbpd_power_monitor_app.c / .h* | Power monitor module entry-point to implement callbacks and application strategy to manage power, faults or critical conditions |
| *usbpd_power_sharing_app.c / .h* | Power sharing module entry-point to implement callbacks and configuration |
| *tracer_emb_conf.h* | Embedded tracer configuration file |

#### 3.1.2.1 *USBPD DPM User*

The USBPD DPM User includes:

1. functions called from the USB PD stack, used to configure and notify USB Type-C and Power Delivery events (see Table 3);
2. miscellaneous callbacks and functions, containing service functions and generic callbacks available as USBPD DPM APIs;
3. wrapper to PE messaging functions, which a set of functions allowing the user application to send specific USB PD control or data messages to the port partner (i.e. `USBPD_DPM_RequestGotoMin` to go to the minimum to pair port).

**Table 3. Relevant USBPD_DPM_User functions**

| Function name | Description |
|---|---|
| `USBPD_DPM_UserInit` | DPM user initialization |
| `USBPD_DPM_UserExecute` | DPM user task to manage alerts |
| `USBPD_DPM_UserCableDetection` | Cable notification handler, managing attach/detach, cable type discovery |
| `USBPD_DPM_UserTimerCounter` | DPM timing management |
| `USBPD_DPM_WaitForTime` | Implementation of the delay used in the USB PD core; it uses the `osDelay` provided by the CMSIS |
| `USBPD_DPM_SetupNewPower` | Interface for power requests coming from USB PD core |
| `USBPD_DPM_HardReset` | Hard reset state machine callback implementation, used to manage the BUS and related messaging during the HR procedure |
| `USBPD_DPM_Notification` | Callback to handle the notification provided by the PE (i.e. when an explicit contract is reached) |
| `USBPD_DPM_GetDataInfo` | To retrieve DPM data/configuration |
| `USBPD_DPM_SetDataInfo` | To set DPM data/configuration |
| `USBPD_DPM_EvaluateRequest` | Ensure local policy evaluating requests from the Sink partner that can be rejected or accepted |
| `USBPD_DPM_EnterErrorRecovery` | DPM callback to allow PE to enter `ERROR_RECOVERY` state |
| `USBPD_DPM_RequestHardReset` | Request the PE to send a hard reset |
| `USBPD_DPM_RequestGotoMin` | Request the PE to send a `GO_TO_MIN` message |
| `USBPD_DPM_RequestMessageRequest` | Request the PE to send a request message |
| `USBPD_DPM_RequestGetSinkCapability` | Request the PE to send a `GET_SINK_CAPABILITY` message |
| `DPM_TurnOffPower` | To turn the power off |
| `DPM_TurnOnPower` | To turn the power on |

#### 3.1.2.2 *USBPD Power Monitor*

The power monitor application defines the callbacks necessary in the Power Monitor module. All static functions are stored in a custom array file and passed to the module during the initialization.

```
PM_Callback_Typedef PM_Callbacks = {
                        .PM_ReadData = PM_ReadData_Handler,
                        .PM_NotifyData = PM_NotifyData_Handler,
                        .PM_CheckStatus = PM_CheckStatus_Handler,
                        .PM_FaultCondition = PM_FaultCondition_Handler,
                        .PM_CriticalCondition = PM_CriticalCondition_Handler,
};
```

The first two functions (`PM_ReadData_Handler` and `PM_NotifyData_Handler`) are related to the Control Task. The other functions are used for Monitor Task which checks the system status, receives events (with minimum latency) and notifies fault and critical conditions at user level. In this module, no direct action on the power is performed.

**Table 4. Power Monitor component files**

| Task | Callback | Description |
|---|---|---|
| **Control Task** | `PM_ReadData_Handler` | Reads $V_{BUS}$ and $I_{BUS}$ data |
| | `PM_NotifyData_Handler` | Notifies the data to the module (called after reading) |
| **Monitor Task** | `PM_CheckStatus_Handler` | Checks the status callback and the whole system, implements the overcurrent/ PGood control strategy and returns ERR or OK. This callback is designed to perform periodic actions |
| | `PM_FaultCondition_Handler` | Fault condition callback, called in case of ERR and to solve and manage faults. If this function returns ERR, the critical condition is notified. The port is moved into safe mode (no $V_{BUS}$) and the USB PD connection is reset |
| | `PM_CriticalCondition_Handler` | Critical condition callback to put the port in safe mode and require a power cycle to start again |

### 3.1.3 Vendor information file (VIF)

The vendor information file (VIF) used to test the solution has been also included in the application folder.

The file contains all the board setup information to correctly run the test activity with USB-IF compliance test tools.

VIF has been included in the software package to ease testing the solution for further customization.

## 3.2 Drivers

The Drivers folder includes:

- the board support package (BSP) which includes the STPD01 and TCPP02-M18 component source codes, as well as the NUCLEO-G071RB and STEVAL-2STPD01 BSP files;
- the Cortex Microcontroller Software Interface Standard (CMSIS) containing the STM32G071RB microcontroller start-up file;
- the STM32G0xx Hardware Abstraction Layer (HAL) that includes the driver to configure and manage the peripheral devices such as I/O ports, interrupts, timers and communication.

### 3.2.1 BSP

Each module of the BSP manages a specific feature of the system, creating a software abstraction of the available hardware.

#### 3.2.1.1 *Component files*

This sub-folder contains the component drivers of STPD01 and TCPP02-M18 devices

**Table 5. STPD01 component files**

| File name | Description |
|---|---|
| *stpd01.c / .h* | STPD01 device abstraction layer |
| *stpd01_reg.c / .h* | STPD01 I²C register communication layer |

**Table 6. TCPP02-M18 component files**

| File name | Description |
|---|---|
| *tcpp0203.c / .h* | TCPP02-M18 device abstraction layer |
| *tcpp0203_reg.c / .h* | TCPP02-M18 I²C register communication layer |

*Note: The component files described in Table 6 also contains the layers for the TCPP03-M20 device which is not embedded in the STEVAL-2STPD01 kit.*

#### 3.2.1.2 *STM32G0xx_NUCLEO file*

The stm32g0xx_nucleo.c file contains a set of other minor firmware functions not directly connected to the proposed application example.

#### 3.2.1.3 *STEVAL-2STPD01 files*

The table below lists the main configuration files constituting the abstraction layer of the expansion board hardware.

**Table 7. STEVAL-2STPD01 expansion board component files**

| File name | Description |
|---|---|
| *steval-2stpd01.c / .h* | Main entry file and common definition, used to initialize the BSP layer |
| *steval-2stpd01_bus.c / .h* | BUS hardware resources |
| *steval-2stpd01_conf.h* | Pin-out configuration and peripheral identification for the main file (USBC peripheral, GPIO used) |
| *steval-2stpd01_errno.h* | Definition of the error types |
| *steval-2stpd01_stpd01.c / .h* | API functions to initialize and manage both STPD01 components |
| *steval-2stpd01_stpd01_conf.h* | Pin-out configuration and peripheral identification for the main file: the STPD01 communication peripheral and the GPIO pins used |
| *steval-2stpd01_tcpp02.c / .h* | API functions to initialize and manage both TCPP02-M18 components |
| *steval-2stpd01_tcpp02_conf.h* | Pin-out configuration and peripheral identification for the main file: the TCPP02-M18 communication peripheral and the GPIO pins used |
| *steval-2stpd01_usbpd_pwr.c / .h* | Used for power management and split in four sections: $V_{BUS}$, $V_{CONN}$, Monitor and Protection |

### 3.2.2 CMSIS

The Cortex Microcontroller Software Interface Standard (CMSIS) driver library contained in this directory is the hardware abstraction layer for microcontrollers based on Arm® Cortex® processors. It enables device support and software interfaces for the processor and its peripherals.

For this software package, CMSIS provides the RTOS services as wrapper of FreeRTOS.

**RELATED LINKS**

*For further details on the CMSIS software packs available in STM32CubeMX, refer to UM1718: STM32CubeMX for STM32 configuration and initialization C code generation*

### 3.2.3 STM32G071RBT6 hardware abstraction layer (HAL) drivers

The HAL drivers sub-folder contains ready-to-use APIs that simplify the user application implementation for a set of STM32 peripherals.

The STM32G071RBT6 HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks).

**RELATED LINKS**

*For further details on STM32G0 HAL drivers, refer to UM2319: Description of STM32G0 HAL and low-layer drivers*

## 3.3 Middleware

### 3.3.1 USB-PD library

USB-PD library is the Middleware stack hosted in the STM32CubeG0 MCU expansion package. It consists of libraries, drivers, sources, APIs and application examples running on many STM32 32-bit microcontrollers.

**RELATED LINKS**

*For further details on STM32G0 microcontroller USB PD, refer to UM2552:" Managing USB power delivery systems with STM32 microcontrollers"*

### 3.3.2 FreeRTOS

FreeRTOS is a widely known real-time operating system (RTOS) for microcontrollers and small microprocessors. It offers many APIs to work with tasks, queues, semaphores, etc. as well as scheduler functions.

The application is set on several tasks defined in different layers that perform specific functions: at USB PD stack level, the most relevant tasks are Type-C cable detection and Policy Engine states machine management.

A further task has been created to manage alerts at DPM level.

Other tasks are defined in the Power Monitor module to implement system monitoring mechanism.

## 3.4 Utilities

The Utilities folder contains the Embedded Tracer, the GUI interface, the Power Monitor and the Power Sharing modules.

### 3.4.1 Embedded tracer

This module implements the embedded tracer based on the USB PD stack library.

**Table 8. Embedded tracer files**

| File name | Description |
|---|---|
| *tracer_emb.c / .h* | Logical implementation |
| *tracer_emb_hw.c / .h* | Hardware support |

**RELATED LINKS**

*UM2552:" Managing USB power delivery systems with STM32 microcontrollers"*

*Visit the wiki page for relevant guides and resources regarding USB Power Delivery*

### 3.4.2 GUI interface

This module includes the APIs related to the GUI used for the STM32CubeMonUCPD and allows logging and tracking the board data output on the GUI, but also to configure the input power supply data in the application setting parameters.

### 3.4.3 Power monitor

**Table 9. Power monitor files**

| File name | Description |
|---|---|
| *usbpd_power_monitor.c / .h* | Power monitor module |

This FreeRTOS-based module is contained in the usbpd_power_monitor file pair. It monitors power and is connected to other layers through callbacks.

Users can customize the actions performed using usbpd_power_monitor_app file pair.

There are two main tasks running:

- `PM_Control_Task`: a high priority task that reads data operation (`PM_ReadData_Handler` callback ) and, consequently, notifies the data collected (`PM_NotifyData_Handler` callback).

  The data collected are the BUS voltage and current; they are stored into the dedicated fields of the `PM_Handle` (ready to be shared with the other part of the system).
- `PM_Monitor_Task`: an event-driven action task with a timeout to periodically perform check and maintenance actions, to get USB PD status notifications (i.e., attach/detach events, explicit contract) or faults (i.e., overcurrent, overvoltage).

When running periodically, it recalls the `PM_CheckStatus_Handler` external function to perform a system check: in case of error, a fault event is generated.

### 3.4.4 Power sharing

The power sharing module is included as compiled library and contains a specific algorithm able to acquire the power supply input settings, calculate the power rate available for each port and dynamically adapt the PDOs to expose, when a power negotiation starts or a generic USB PD event occurs.

**Table 10. Power sharing files**

| File name | Description |
|---|---|
| *usbpd_power_sharing .h* | Power sharing module header |

## 3.5 Libraries

The compiled libraries included in the software package represents two of the ST IPs featuring the STEVAL-2STPD01 solution:

- **USBPD Core Library** (available in the STM32CubeG0 package): hosting all the functions related to the USB-PD Middleware stack USB-PD Policy engine and Protocol layer.

  Path→ `$ROOT\Firmware\Middlewares\ST\STM32_USBPD_Library\Core\lib`

  – `USBPDCORE_PD3_FULL_CM0PLUS_wc32.a` → STM32CubeIDE and EWARM (wc32)

  – `USBPDCORE_PD3_FULL_CM0PLUS_Keil.lib` → µVision
- **Power Sharing Library**:

  Path→`$ROOT\Firmware\Utilities\PowerSharing\lib`:

  – `USBPD_PowerSharing.a` → STM32CubeIDE and EWARM (wc32)

  – `USBPD_PowerSharing_Keil.lib` → µVision

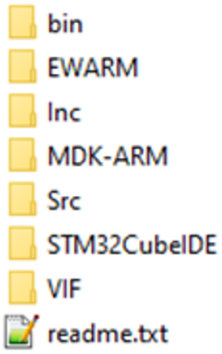
# 4 Workspaces

To customize and debug the application, the package supports three integrated development environments (IDEs): STMicroelectronics STM32CubeIDE, IAR EWARM and Keil µVision / MDK-ARM.

The project files are located in the application folder as shown below.

**Figure 4. IDE folders**

STM32CubeIDE– STMicroelectronics
EWARM – IAR
µVision / MDK-ARM – Keil




## 4.1 STM32CubeIDE

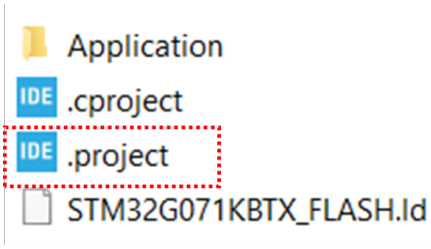
STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors.

It is based on the Eclipse®/CDT framework and GCC toolchain for the development and GDB for the debugging.

To open the project, select the STM32CubeIDE folder and open the *.project* file.

The path is: `$ROOT\Firmware\Projects\STEVAL-2STPD01\STM32CubeIDE`

**Figure 5. STM32CubeIDE project file**

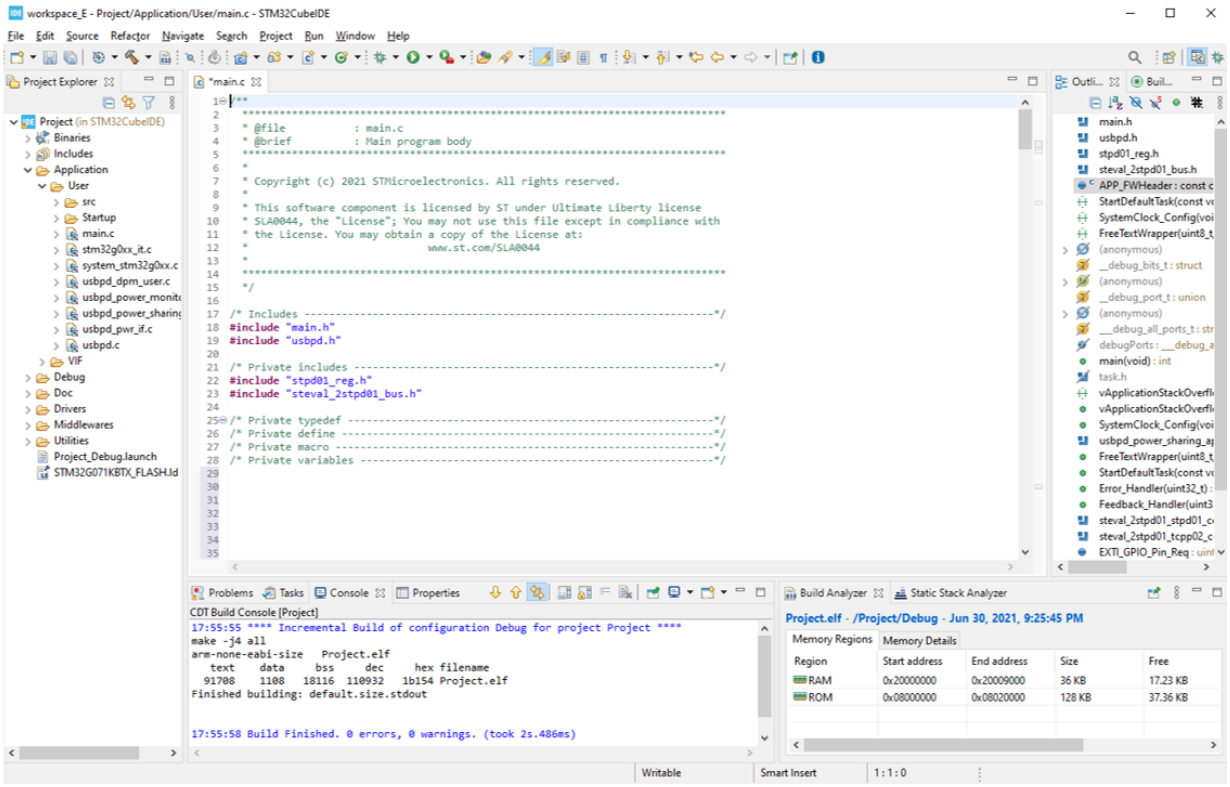



To open the STM32CubeIDE project, you can:

1. open the environment and select the STM32CubeIDE project folder in the [**File**]>[**Open Projects from File System**] menu;
2. or open the project by double clicking it in the folder: when the project is successfully imported, a notification pops up.

At the end of the import, close the [**Information Center**] tab and expand the workspace tab to view the complete folder structure. In both cases, the IDE is ready.

**Figure 6. Project opened in the STM32CubeIDE environment**



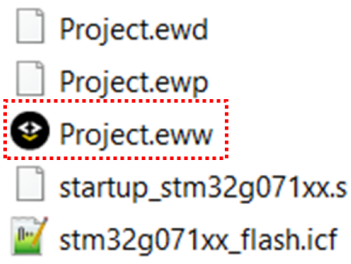

*Note: The project has been tested with STM32CubeIDE v1.6.1.*

## 4.2 EWARM - IAR

IAR Embedded Workbench is compliant with Arm embedded application binary interface (EABI) and Arm Cortex microcontroller software interface standard (CMSIS).

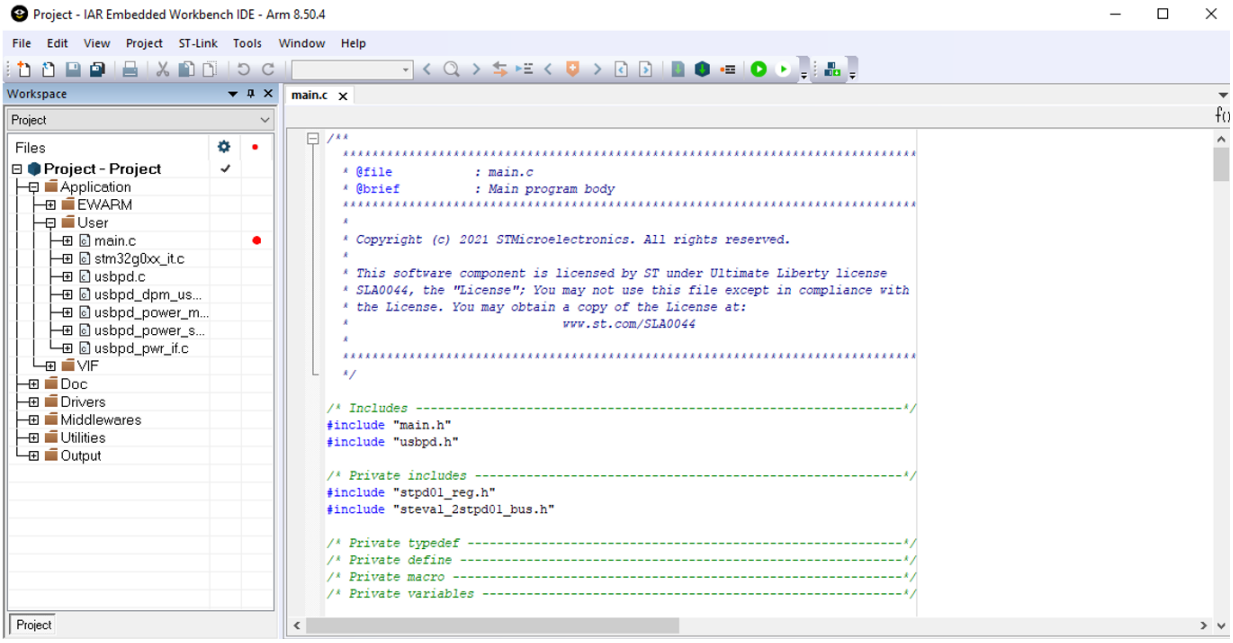
The path is: `$ROOT\Firmware\Projects\STEVAL-2STPD01\EWARM`

**Figure 7. EWARM project file**




To start the evaluation, double-click the *Project.eww* file and open it.

**Figure 8. Project opened in the EWARM environment**



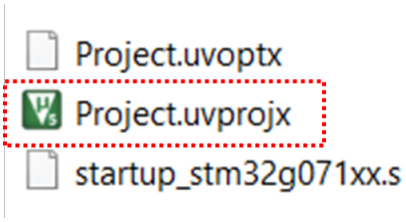

*Note: The project was tested with EWARM v8.50.x. If there are several EWARM versions in your PC, open the correct IDE version and select the Project.eww file from the **[menu]>[open workspace]**.*

## 4.3 µVision/MDK-ARM - Keil

The µVision IDE and debugger is developed by Keil and supports the user in the development and debugging.

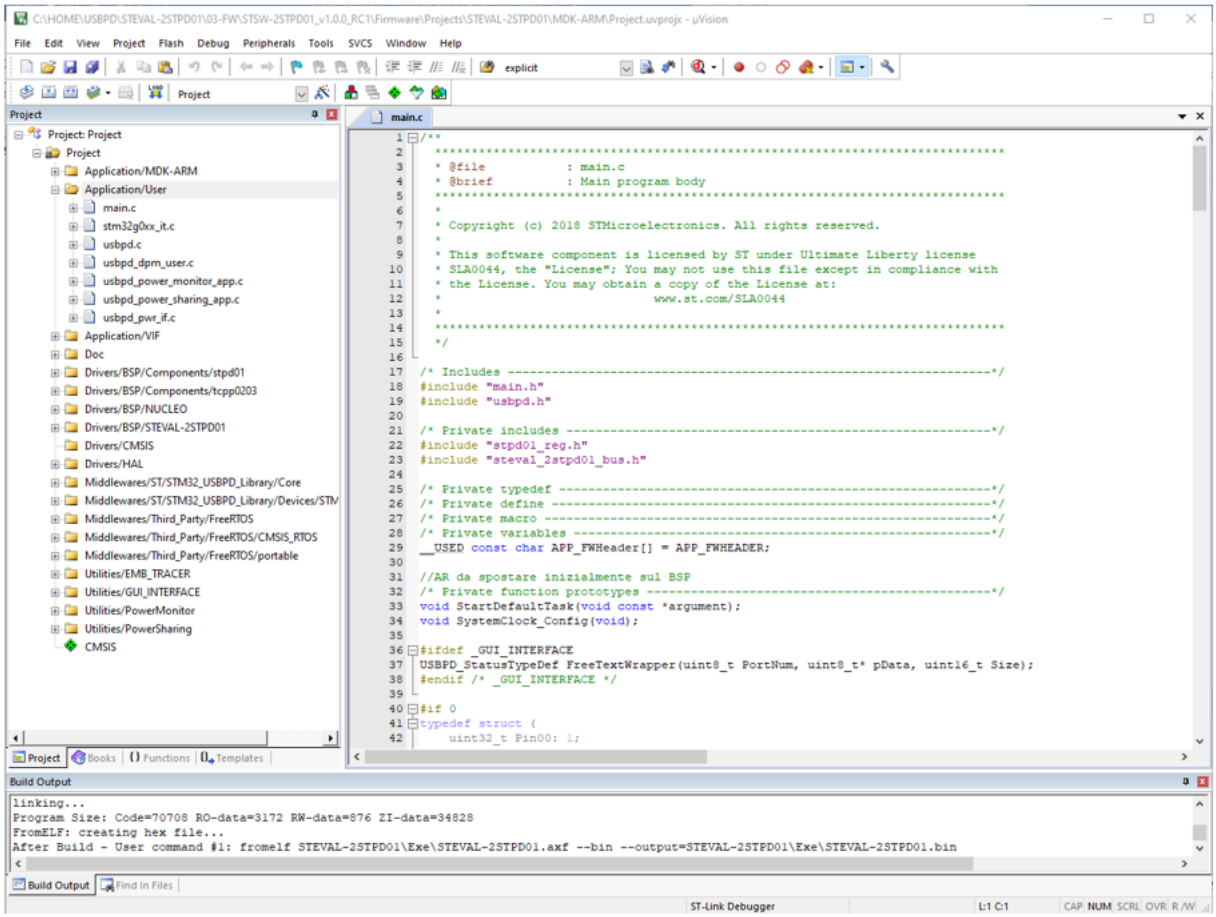
The path is: `$ROOT\Firmware\Projects\STEVAL-2STPD01\MDK-ARM`

**Figure 9. µVision MDK-ARM project file**




To start working with this development environment, double-click the *Project.uvprojx* file and open it.

**Figure 10. Project opened in the µVision MDK-ARM environment**




*Note: The project has been tested with µVision v5.34.0.0 and MDK-ARM 5.34.*

# 5 Licensing information

STSW-2STPD01 is delivered under the *Mix Ultimate Liberty+OSS+3rd-party V1 license*.

The software components provided within this package come with different license agreements as listed in the following table.

**Table 11. Software component license agreements**

| Software component | Owner | License |
|---|---|---|
| Cortex®-M CMSIS | Arm® | BSD 3-Clause |
| FreeRTOS™ Kernel | Copyright(C) 2017 Amazon.com, Inc. or its affiliates | MIT open source license |
| STM32G0 HAL/LL APIs | STMicroelectronics International N.V. | BSD 3-Clause |
| STM32 USB-PD Library | STMicroelectronics International N.V. | Ultimate Liberty software license agreement (SLA0044) |
| STSW-2STPD01 | STMicroelectronics International N.V. | Software package license agreement (SLA0048) |
| STSW-2STPD01 BSP APIs | STMicroelectronics International N.V. | Ultimate Liberty software license agreement (SLA0044) |
| STSW-2STPD01 Power Sharing Library | STMicroelectronics International N.V. | Ultimate Liberty software license agreement (SLA0044) |

# Revision history

Table 12. **Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 22-Nov-2021 | 1 | Initial release. |

# Contents

# List of tables

# List of figures