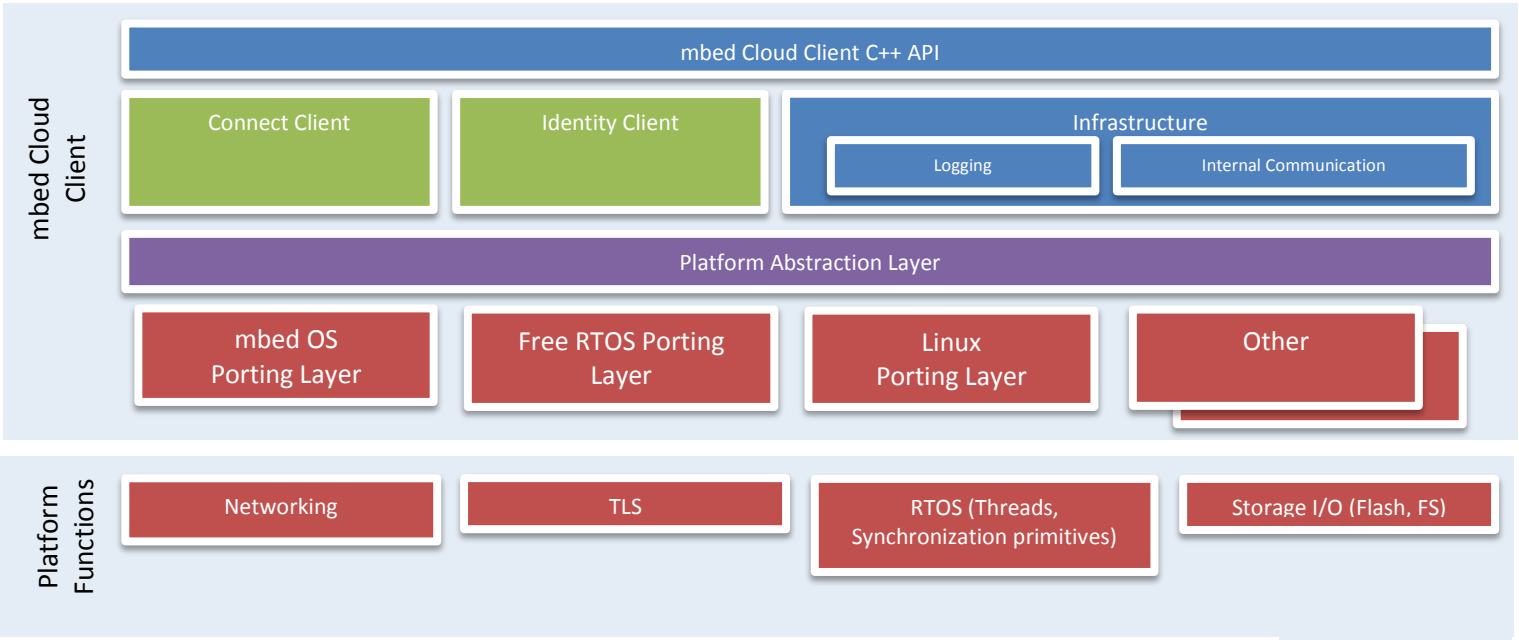
PAL Porting Guide

• PAL Introduction

A Platform Abstraction Layer connects the mbed-client with the underlying platform.



The main purpose of PAL is to enable easy and fast mbed client services portability. Allowing them to operate over wide range of Cortex-M based platforms running different operating systems with various libraries (Networking for example).

Note: Currently mbedTLS library is a requirement from the underlying platform to support mbed client porting.

• PAL Layers

1. Services APIs Layer

These APIs are exposed to the mbed client services;
Implemented by the PAL team.
Customer should not deal with those APIs.

2. Platform APIs Layer

This layer provides a standard set of base line requirements from the platform side that should be implemented in order to make the mbed client ready for running on the target platform.

This layer is implemented differently for each OS or library; **PAL provides a reference implementation over mbedOS which was developed and tested by the PAL team.**

Note: for detailed Platform API documentation, please read the PAL.chm file under PAL/Docs/

• PAL Modules

1. Networking

The Networking stack provides a POSIX like abstraction layer which provides access to TCP and UDP based communication using sockets.

Platform layer should implement synchronous and asynchronous sockets API.

The APIs which MUST be implemented by the platform exists in:

PAL/Source/Port/Platform-API/pal_plat_network.h

mbedOS Reference implementation can be found in:

PAL/Source/Port/Reference-Impl/mbedOS/Networking/pal_plat_network.cpp

2. RTOS: (real-time operating-system)

RTOS module is responsible for providing RTOS functionality; which include primitives such as: (Threads, Mutexes and Semaphores) also it provides an API for kernel ticks, timers, memory-pool and message-queue.

The APIs which MUST be implemented by the platform exists in:

PAL/Source/Port/Platform-API/pal_plat_rtos.h.

Reference implementation can be found in:

PAL/Source/Port/Reference-Impl/mbedOS/RTOS/pal_plat_rtos.c

• How to Build PAL

TBD

• How to port PAL for new platform

In order to port for new platform, customer needs to implement the "Platform-API" interfaces (or use one of the existing reference implementations, if applicable).
It is important to add the customer files to the relevant makefile.

○ Modules Porting Notes

a. RTOS

The major assumption exists in PAL design that each thread has unique priority and the number of threads is limited by a defined value "**PAL_MAX_NUMBER_OF_THREADS**" (defined in PAL/Source/PAL-Impl/Services-API/pal_configuration.h).

b. Networking

The Networking module has three different feature sets that can be selected at compile time:

- TCP socket support
- Asynchronous socket support
- DNS name resolution support

Please implement all the three sets for full service functionality.

• PAL Tests

As a part of PAL code, PAL provides a Unity Test based Framework to test the existing APIs. This framework exists under:

PAL/Test/

○ PAL Tests Sources

Pal Tests code exist in: PAL/Test/Unitest/ for each module there is a test file, test runner and test main for specific OS.

○ PAL Tests Build & Run

In order to build PAL test executable, partner need to do the following: (for mbedOS5.1.3 over Freescale-K64F board)

1. Define the environment variable: MBEDOS_ROOT to be the father folder of "mbed-os[1]".

[1] Folder which holds the mbedOS.

2. cd $(PAL_FOLDER)/Test/
3. make mbedOS_all - This will build the tests for mbedOS5.1 over Freescale-K64F board.
4. In order to build and run the tests over the platform please run:

   $ make mbedOS_check

5. In order to see debug prints please send the following flag "DEBUG=1" in compilation command:

   $ make mbedOS_check DEBUG=1

6. In order to build single module tests please edit "$(PAL_FOLDER)/Test/makefile" under mbedOS platform, please change the value of the "TARGET_CONFIGURATION_DEFINES" to the desired module: (default value is for all exist modules)

   * HAS_RTOS --> RTOS module APIs

   * HAS_SOCKET --> Networking module APIs

   If partner choose to run all tests, the framework will build each modules' tests in separate bin, copy it to the board, run them and collect the results.

Notes:

1. pal_init() API which is called by tests, require network connectivity since it also initialize network module.
2. HAS_RTOS flag compile basic RTOS tests, in order to run tests for Threads, Semaphores and Mutexes and additional flag should be added to the make command:
   a. $ make mbedOS_check PAL_TEST="THREAD_UNITY_TEST"
   b. $ make mbedOS_check PAL_TEST="MUETX_UNITY_TEST"
   c. $ make mbedOS_check PAL_TEST="SEMAPHORE_UNITY_TEST"

○ Expected Tests output

Here are captured results from successful test run:

a. RTOS

```
==== mbedos_pal_rtos ====
-------- -------------------------------- ----
pal_rtos  pal_osKernelSysTick_Unity         PASS
pal_rtos  pal_osKernelSysTick64_Unity       PASS
pal_rtos  pal_osKernelSysTickMicroSec_Unity  PASS
pal_rtos  pal_osKernelSysMilliSecTick_Unity  PASS
pal_rtos  pal_osKernelSysTickFrequency_Unity  PASS
pal_rtos  pal_osDelay_Unity                 PASS
pal_rtos  BasicTimeScenario                 PASS
pal_rtos  TimerUnityTest                    PASS
pal_rtos  MemoryPoolUnityTest               PASS
pal_rtos  MessageUnityTest                  PASS
pal_rtos  AtomicIncrementUnityTest          PASS
pal_rtos  PrimitivesUnityTest1              PASS
pal_rtos  PrimitivesUnityTest2              PASS
-------- -------------------------------- ----
 TOTAL   PASS   FAIL   IGNORE
------- ------ ------ --------
   13    13     0      0
```

Final status: PASS.

b. Network

```
==== mbedos_pal_socket ====
---------- -------------------------- ----
pal_socket  socketUDPCreationOptionsTest  PASS
pal_socket  basicTCPclinetSendRecieve     PASS
pal_socket  basicUDPclinetSendRecieve     PASS
pal_socket  basicSocketScenario3          PASS
pal_socket  basicSocketScenario4          PASS
---------- -------------------------- ----
 TOTAL   PASS   FAIL   IGNORE
------- ------ ------ --------
   5     5      0      0
```

Final status: PASS.