# Separating Vendor Code from TF-M Source Tree

Roman Mazurak
2021.10.08

## Platforms code & TF-M project challenges

- Negative experience for TF-M users.
- Problems for platforms vendors.
- Difficulties to evolve TF-M.

## Negative experience for TF-M users

- The large amount of sources complicates the development, as they affect the IDE, build time and, testing process.
- It's much harder to understand TF-M code dependencies because there are lot of implementations for the same functions across different platforms folders.
- It's hard to understand why TF-M source code contains a lot of useless code from the end user point of view.

## Problems for platforms vendors

- It is unlikely to build a quality testing process on the basis of a common repository for all platforms. There are too many factors and tests that need to be validated for each platform. And it is unrealistic to organize strong validation for all platforms.
- The good support of the platform by the vendor requires the creation of their own test benches on real hardware. And some of such tests can't be published and used during validation of patches applied by the community.
- It's impossible to support platform for outdated versions of TF-M in the common repository. Because it requires to change branches that was already released.

## Difficulties to evolve TF-M

- Any change to the TF-M that affects platforms in addition to approval also requires implementation in the platform code. This negatively affects the development process, as it requires human resources. And it also significantly affects the ability to release TF-M when there are some platforms without all necessary patches.
- It's clear that TF-M core should provide proven security architecture. And platform vendors should be responsible for the appropriate implementation of their code base. But currently it's hard to say who is responsible in the situation when there is a security breach or an untested code in some platform code.

## Other projects experience

*Linux Kernel and drivers.*

- There are two types of drivers for Linux. First kind is part of the Linux Kernel source tree while second one are maintained as a separate projects.
- Using the driver that comes with the kernel does not guarantee that it will compile and run without issues. It's possible that the driver code become broken or not updated to the latest API. So, from the user perspective it's hard to tell that some specific driver in Linux will work for the project. It's happen that selection of the working combination of Linux Kernel and set of drivers is a real challenge for the developers.
- On the other side it’s much easier to use drivers which are maintained outside of the source tree. Because such kind of drivers are tested much better by teams that own drivers projects. Errata and revision information are much up to date. This helps to make a decision using less efforts.

## Steps to move forward

- What should we do with existing platforms that are part of TF-M source tree?
- TF-M related migration tasks.
- How to bind TF-M Core and vendor platform code in development process?
- How to organize platform development infrastructure?
- What should be decided by TF-M users?

## What should we do with existing platforms that are part of TF-M source tree?

- Platforms that fall under the deprecation will be removed from the TF-M repository in natural way.
- What should we do with platforms which are not updated according to new TF-M requirements and APIs? What should we do when TF-M should be released but some platforms are not up to date? Do we need to remove such platforms and ask platform owners to publish them as a separate code?
- The most radical way is to forcibly remove all platforms except reference one. But its significant drawback is that the development is carried out by many people and they (as well as git) count on the presence of branches and sources in a certain place. Therefore, this option is probably best implemented within the vendor's team, taking into account internal processes, updating testing environment and other necessary infrastructure, as well as notifying independent contributors when and how this will happen.
- It should be a good idea to recommend vendors to create a migration plans and publish it.

## TF-M related migration tasks

- Do not allow to merge new platforms into **platform/ext/target** folder.
- Create a patch to allow building TF-M via usage of **add_subdirectory**() CMake function. Currently CMake scripts use CMAKE_SOURCE_DIR, CMAKE_BINARY_DIR variables instead of PROJECT_XXX_DIR, CMAKE_CURRENT_XXX_DIR or CMAKE_CURRENT_LIST_DIR.
- Implement TF-M Tests Out-of-Tree Build (work in progress).
- Update build scripts and cmake files which use platform specific features. For example TFM_L3_PLATFORM_LISTS which contains list of platforms that support isolation level 3.
- Publish document which describes migration process.

## How to bind TF-M Core and vendor platform code in development process?

- TF-M Core and vendor platform code depends on each other. TF-M Core can publish API version for each components or only TF-M release version (TFM_VERSION CMake variable).
- Platforms should validate TF-M version and/or TF-M components API version variables during build and generate an error in case of incompatibility.
- It will be up to user to decide how to resolve incompatibility issue. Whether to download the appropriate version of TF-M, or switch to another version of the platform, which is compatible with the existing version of TF-M.
- It’s expected that TF-M version will not be changed until the release, but the actual API will be continuously updated. So, in this case platform code will not be compatible with TF-M. We can observe such kind of problems during builds and test suites runs.

## How to organize platform development infrastructure?

It's also important that vendors should update own development infrastructure which should link together all necessary components like TF-M core, platform code, platform specific additional libraries, test suites and test scripts. There are two variants :

Using git submodule. It’s more convenient for development.

More over vendors can create multiple projects for different purposes like one for platform development another for platform testing.

Using scripts that download all necessary components during building and testing.

This approach can be useful for CI scenarios.

## What should be decided by TF-M users?

The user will receive a flexible tool that will allow easy integration of TF-M together with vendor components into the user build infrastructure.