STM32MP1 Developer Package

STM32MP1 Developer Package

Stable: 23.07.2020 - 14:54 / Revision: 07.07.2020 - 12:09

This article describes how to get and use the **Developer Package** of the **STM32MPU Embedded Software** for any development platform of the **STM32MP1 family** (STM32MP15 boards), in order to modify some of its pieces of software, or to add applications on top of it.

It lists some **prerequisites** in terms of knowledges and development environment, and gives the **step-by-step** approach to download and install the STM32MPU Embedded Software components for this Package.

Finally, it proposes some guidelines to upgrade (add, remove, configure, improve...) any piece of software.

Contents

1 Developer Package content

If you are not yet familiar with the **STM32MPU Embedded Software** distribution and its **Packages**, please read the following articles:

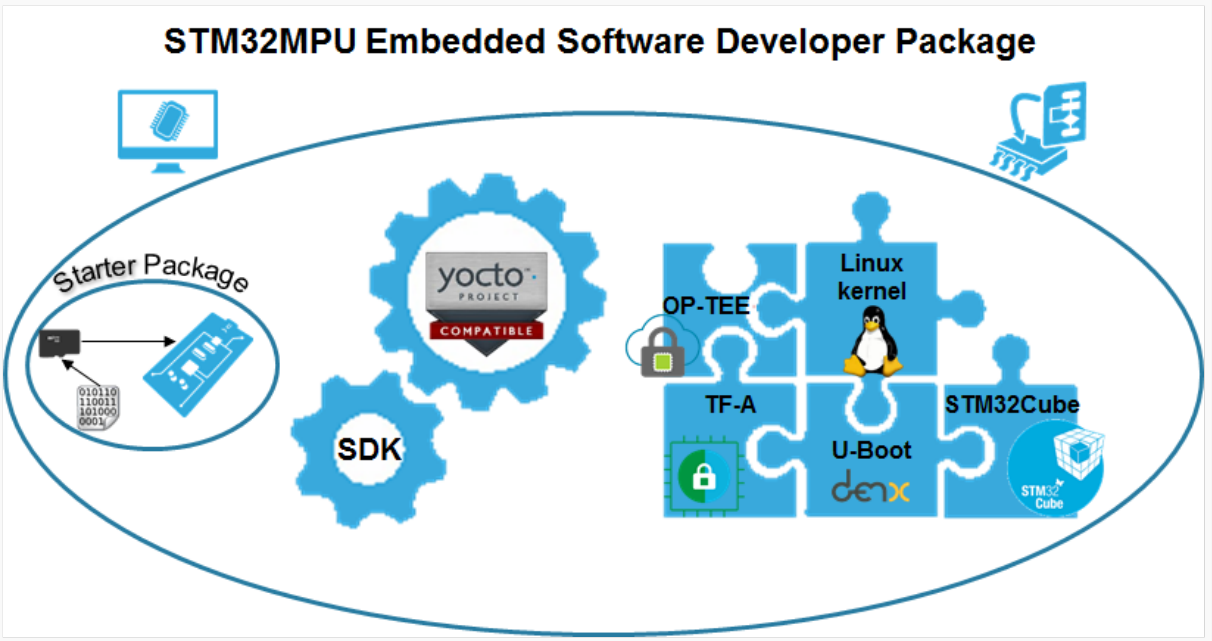
- Which STM32MPU Embedded Software Package better suits your needs (and especially the Developer Package chapter)
- STM32MPU Embedded Software distribution

If you are already familiar with the Developer Package for the STM32MPU Embedded Software distribution, the fast links to essential commands might interest you.

To sum up, this **Developer Package** provides:

- for the **OpenSTLinux distribution** (development on Arm® Cortex®-A processor):
  - the **software development kit** (SDK), based on Yocto SDK, for cross-development on an host PC
  - the following pieces of software in **source code**:
    - Linux® kernel
    - U-Boot
    - Trusted Firmware-A (TF-A)
    - optionally, Open source Trusted Execution Environment (OP-TEE)
  - the **debug symbol files** for Linux® kernel, U-Boot and TF-A
- for the **STM32Cube MPU Package** (development on Arm® Cortex®-M processor):
  - all pieces of software (BSP, HAL, middlewares and applications) in **source code**
  - the **integrated development environment (IDE)** (STM32-CoPro-MPU Eclipse plugin)

Note that, the application frameworks for the OpenSTLinux distribution are not available as source code in this Package.



Components of the Developer Package

2 Developer Package step-by-step overview

The steps to get the STM32MPU Embedded Software Developer Package ready for your developments, are:

Checking the prerequisites

Installing the Starter Package for your board

Installing the components to develop software running on Arm® Cortex®-A (OpenSTLinux distribution)

- ☐ Installing the SDK (**mandatory** for any development on Arm® Cortex®-A)
- ☐ Installing the Linux kernel (**mandatory only** if you plan to modify the Linux kernel or to add external out-of-tree Linux kernel modules)
- ☐ Installing the U-Boot (**mandatory only** if you plan to modify the U-Boot)
- ☐ Installing the TF-A (**mandatory only** if you plan to modify the TF-A)
- ☐ Installing the debug symbol files (**mandatory only** if you plan to debug Linux® kernel, U-Boot or TF-A with GDB)

Installing the components to develop software running Arm Cortex-M (STM32Cube MPU Package)

- ☐ Installing the Eclipse IDE (**mandatory** for any development on Arm® Cortex®-M)
- ☐ Installing the STM32Cube MPU Package (**mandatory only** if you plan to modify the Cube firmware)

Once these steps are achieved, you are able to:

- develop software running on Arm Cortex-A
  - Modifying the Linux kernel
  - Adding external out-of-tree Linux kernel modules
  - Adding Linux user space applications
  - Modifying the U-Boot
  - Modifying the TF-A
- develop software running on Arm Cortex-M4

3 Checking the prerequisites

3.1 Knowledges

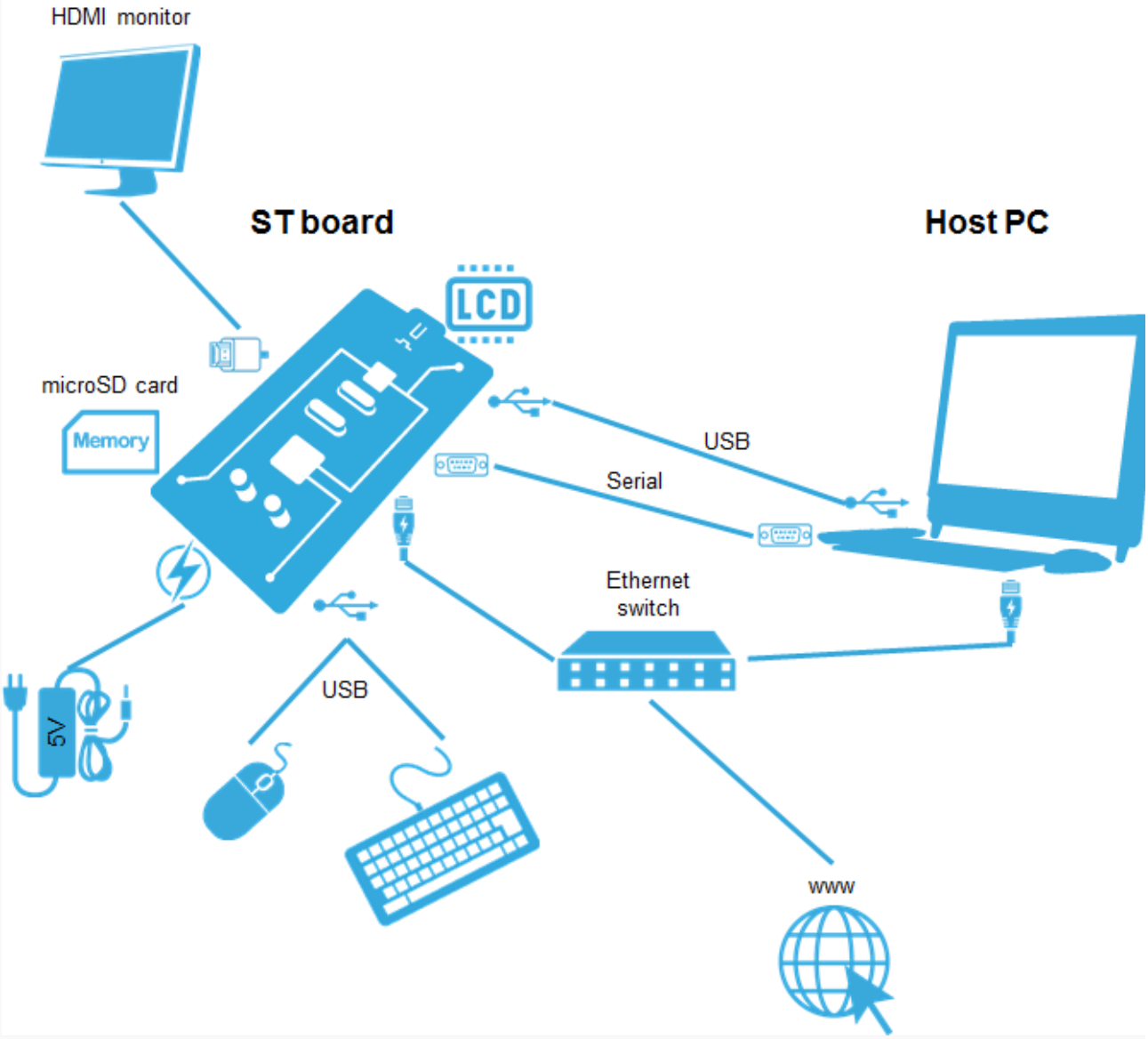
The STM32MP1 Developer Package aims at enriching a Linux-based software for the targeted product: basic knowledges about Linux are recommended to make the most of this Package.

Having a look at the STM32MPU Embedded Software architecture overview is also highly recommended.

3.2 Development setup

The recommended setup for the development PC (host) is specified in the following article: PC prerequisites.

Whatever the development platform (board) and development PC (host) used, the range of possible development setups is illustrated by the picture below.



Development setup for Developer and Distribution Packages

The following components are **mandatory**:

- Host PC for cross-compilation and cross-debugging, installed as specified above
- Board assembled and configured as specified in the associated Starter Package article
- Mass storage device (for example, microSD card) to load and update the software images (binaries)

The following components are **optional**, but **recommended**:

- A serial link between the host PC (through Terminal program) and the board for traces (even early boot traces), and access to the board from the remote PC (command lines)
- An Ethernet link between the host PC and the board for cross-development and cross-debugging through a local network. This is an alternative or a complement to the serial (or USB) link
- A display connected to the board, depending on the technologies available on the board: DSI LCD display, HDMI monitor (or TV) and so on
- A mouse and a keyboard connected through USB ports

Additional optional components can be added by means of the connectivity capabilities of the board: cameras, displays, JTAG, sensors, actuators, and much more.

4 Installing the Starter Package

Before developing with the Developer Package, **it is essential to start up your board thanks to its Starter Package**. All articles relative to Starter Packages are found in Category:Starter Package: find the one that corresponds to your board, and follow the installation instructions (if not yet done), before going further.

In brief, it means that:

- your board boots successfully
- the flashed image comes from the same release of the STM32MPU Embedded Software distribution than the components that will be downloaded in this article

Thanks to the Starter Package, **all Flash partitions are populated**.

Then, with the Developer Package, it is possible to modify or to upgrade the partitions independently one from the others.

For example, if you only want to modify the Linux kernel (part of *bootfs* partition), installing the SDK and the Linux kernel are enough; no need to install anything else.

5 Installing the components to develop software running on Arm Cortex-A (OpenSTLinux distribution)

5.1 Installing the SDK

Optional step: it is mandatory only if you want to modify or add software running on Arm Cortex-A (e.g. Linux kernel, Linux user space applications...).

The SDK for OpenSTLinux distribution provides a stand-alone cross-development toolchain and libraries tailored to the contents of the specific image flashed in the board. If you want to know more about this SDK, please read the SDK for OpenSTLinux distribution article.

- The STM32MP1 SDK is delivered through a tarball file named : **en.SDK-x86_64-stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24.tar.xz**
- Download and install the STM32MP1 SDK.

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [1]. If, trying to download, you encounter a "403 error", you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

<table>
<tr><th></th><th>STM32MP1 Developer Package SDK - STM32MP15-Ecosystem-v2.0.0 release</th></tr>
<tr><td>Download</td><td>You need to be logged on my.st.com before accessing the following link:
en.SDK-x86_64-stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24.tar.xz</td></tr>
<tr><td>Installation</td><td>
Uncompress the tarball file to get the SDK installation script
<pre>tar xvf en.SDK-x86_64-stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24.tar.xz</pre>
If needed, change the permissions on the SDK installation script so that it is executable
<pre>$ chmod +x stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-openstlinux-5.4-dunfell-mp1-20-06-24.sh</pre>
Run the SDK installation script
Use the -d <SDK installation directory absolute path> option to specify the absolute path to the directory in which you want to install the SDK (<SDK installation directory>)
If you follow the proposition to organize the working directory, it means:
<pre>$ ./stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-openstlinux-5.4-dunfell-mp1-20-06-24.sh -d <working directory absolute path>/Developer-Package/SDK</pre>
A successful installation outputs the following log:
<pre>ST OpenSTLinux - Weston - (A Yocto Project Based Distro) SDK installer version 3.1-snapshot
==
You are about to install the SDK to "<working directory absolute path>/Developer-Package/SDK". Proceed [Y/n]?
Extracting SDK...
....
....
.....................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
 $. <working directory absolute path>/Developer-Package/SDK/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi</pre>
</td></tr>
<tr><td>Release</td><td>Details about the content of the SDK are available in the associated STM32MP15 ecosystem release note.</td></tr>
</table>

	STM32MP1 Developer Package SDK - STM32MP15-Ecosystem-v2.0.0 release
note	If you are interested in older releases, please have a look into the section Archives.

- The SDK is in the *<SDK installation directory>*:

```
<SDK installation directory>                                                    SDK
for OpenSTLinux distribution: details in Standard SDK directory structure article
├── environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi  Environment setup
script for Developer Package
├── site-config-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
├── sysroots
│   ├── cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi                Target sysroot
(libraries, headers, and symbols)
│   │   └── [...]
│   └── x86_64-ostl_sdk-linux                                     Native sysroot
(libraries, headers, and symbols)
│       └── [...]
└── version-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

Now that the SDK is installed, please do not move or rename the *<SDK installation directory>*.

5.1.1 Starting up the SDK

The SDK environment setup script must be run once in each new working terminal in which you cross-compile:

```
PC $> source <SDK installation directory>/environment-setup-cortexa7hf-neon-vfpv4-ostl-
linux-gnueabi
```

The following checkings allow to ensure that the environment is correctly setup:

- Check the target architecture

```
PC $> echo $ARCH
arm
```

- Check the toolchain binary prefix for the target tools

```
PC $> echo $CROSS_COMPILE
arm-ostl-linux-gnueabi-
```

- Check the C compiler version

```
PC $> $CC --version
arm-ostl-linux-gnueabi-gcc (GCC) <GCC version>
[...]
```

- Check that the SDK version is the expected one

```
PC $> echo $OECORE_SDK_VERSION
<expected SDK version>
```

If any of these commands fails or does not return the expected result, please try to reinstall the SDK.

5.2 Installing the Linux kernel

Optional step: it is mandatory only if you want to modify the Linux kernel (configuration, device tree, driver...), or to add external out-of-tree Linux kernel modules.

Prerequite: the SDK is installed.

5.2.1 Downloading the Linux kernel

- The STM32MP1 Linux kernel is delivered through a tarball file named **en.SOURCES-kernel-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 Linux kernel

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [2]. If, trying to download, you encounter a "403 error", you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package Linux kernel - STM32MP15-Ecosystem-v2.0.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-kernel-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the Linux kernel (Linux kernel source code, ST patches, ST configuration fragments...):
	`PC $> $ tar xvf en.SOURCES-kernel-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz`
	`PC $> $ cd stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24/sources/arm-ostl-linux-gnueabi/linux-stm32mp-5.4.31-r0`
	`PC $> $ tar xvf linux-5.4.31.tar.xz`

	STM32MP1 Developer Package Linux kernel - STM32MP15-Ecosystem-v2.0.0 release
Release note	Details of the content of the Linux kernel are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **Linux kernel installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-20-06-24/sources/arm-ostl-linux-gnueabi* directory, and is named *linux-stm32mp-<kernel version>*:

```
linux-stm32mp-5.4.31-r0     Linux kernel installation directory
├── [*].patch               ST patches to apply during the Linux kernel preparation (see
next chapter)
├── fragment-[*].config     ST configuration fragments to apply during the Linux kernel
configuration (see next chapter)
├── linux-5.4.31            Linux kernel source code directory
├── linux-5.4.31.tar.xz     Tarball file of the Linux kernel source code
├── README.HOW_TO.txt       Helper file for Linux kernel management: reference for Linux
kernel build
└── series                  List of all ST patches to apply
```

5.2.2 Building and deploying the Linux kernel for the first time

It is mandatory to execute once the steps specified below before modifying the Linux kernel, or adding external out-of-tree Linux kernel modules.

The partitions related to the Linux kernel are:

- the *bootfs* partition that contains the Linux kernel U-Boot image (*uImage*) and device tree
- the *rootfs* partition that contains the Linux kernel modules

The Linux kernel might be cross-compiled, either in the source code directory, or in a dedicated directory different from the source code directory.
This last method is recommended as it clearly separates the files generated by the cross-compilation from the source code files.

The README.HOW_TO.txt helper file is **THE** reference for the Linux kernel build

The SDK must be started

Open the *<Linux kernel installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the Linux kernel (optional but recommended)

prepare the Linux kernel (applying the ST patches)

configure the Linux kernel (applying the ST fragments)

cross-compile the Linux kernel

deploy the Linux kernel (i.e. update the software on board)

The Linux kernel is now installed: let's modify the Linux kernel, or add external out-of-tree Linux kernel modules.

5.3 Installing the U-Boot

Optional step: it is mandatory only if you want to modify the U-Boot.

Prerequite: the SDK is installed.

5.3.1 Downloading the U-Boot

- The STM32MP1 U-Boot is delivered through a tarball file named **en.SOURCES-u-boot-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 U-Boot

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [3]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package U-Boot - STM32MP15-Ecosystem-v2.0.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-u-boot-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the U-Boot (U-Boot source code, ST patches and so on):
	`PC $> $ tar xvf en.SOURCES-u-boot-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz`
	`PC $> $ cd stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24/sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-2020.01-r0`
	`PC $> $ tar xvf u-boot-stm32mp-2020.01-r0.tar.gz`

Release note	Details of the content of the U-Boot are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **U-Boot installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-20-06-24/sources/arm-ostl-linux-gnueabi* directory, and is named *u-boot-stm32mp-<U-Boot version>*:

```
u-boot-stm32mp-2020.01-r0
├── [*].patch                          U-Boot installation directory
├── u-boot-stm32mp-2020.01             ST patches to apply during the U-Boot preparation (see next chapter)
├── Makefile.sdk                       U-Boot source code directory
├── README.HOW_TO.txt                  Makefile for the U-Boot compilation
├── series                             Helper file for U-Boot management: reference for U-Boot build
└── u-boot-stm32mp-2020.01-r0.tar.gz   List of all ST patches to apply
                                       Tarball file of the U-Boot source code
```

5.3.2 Building and deploying the U-Boot for the first time

It is mandatory to execute once the steps specified below before modifying the U-Boot.

As explained in the boot chains overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the U-Boot is the *ssbl* one that contains the U-Boot and its device tree blob.

The README.HOW_TO.txt helper file is **THE** reference for the U-Boot build

The SDK must be started

Open the *<U-Boot installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the U-Boot (optional but recommended)

prepare the U-Boot (applying the ST patches)

cross-compile the U-Boot

deploy the U-Boot (i.e. update the software on board)

The U-Boot is now installed: let's modify the U-Boot.

5.4 Installing the TF-A

Optional step: it is mandatory only if you want to modify the TF-A.

Prerequite: the SDK is installed.

5.4.1 Downloading the TF-A

- The STM32MP1 TF-A is delivered through a tarball file named **en.SOURCES-tf-a-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 TF-A

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [4]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package TF-A - STM32MP15-Ecosystem-v2.0.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-tf-a-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz
	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:

	STM32MP1 Developer Package TF-A - STM32MP15-Ecosystem-v2.0.0 release

Installation	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the TF-A (TF-A source code, ST patches...):
	`PC $> $ tar xvf en.SOURCES-tf-a-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz`
	`PC $> $ cd stm32mp1-openstlinux-5.;4-dunfell-mp1-20-06-24/sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-2.2.r1-r0`
	`PC $> $ tar xvf tf-a-stm32mp-2.2.r1-r0.tar.gz`

Release note	Details about the content of the TF-A are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **TF-A installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-20-06-24/sources/arm-ostl-linux-gnueabi* directory, and is named *tf-a-stm32mp-<TF-A version>*:

```
tf-a-stm32mp-2.2.r1-r0                TF-A installation directory
├── [*].patch                         ST patches to apply during the TF-A preparation
(see next chapter)
├── tf-a-stm32mp-2.2.r1               TF-A source code directory
├── Makefile.sdk                      Makefile for the TF-A compilation
├── README.HOW_TO.txt                 Helper file for TF-A management: reference for TF-A
build
├── series                            List of all ST patches to apply
└── tf-a-stm32mp-2.2.r1-r0.tar.gz     Tarball file of the TF-A source code
```

5.4.2 Building and deploying the TF-A for the first time

It is mandatory to execute once the steps specified below before modifying the TF-A.

As explained in the boot chains overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the TF-A is the *fsbl* one.

The README_HOWTO.txt helper file is **THE** reference for the TF-A build

The SDK must be started

Open the *<TF-A installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the TF-A (optional but recommended)

prepare the TF-A (applying the ST patches)

cross-compile the TF-A

deploy the TF-A (i.e. update the software on board)

The TF-A is now installed: let's modify the TF-A.

5.5 Installing the OP-TEE

Optional step: it is mandatory only if you want to modify the OP-TEE.

Prerequite: the SDK is installed.

5.5.1 Downloading the OP-TEE

- The STM32MP1 OP-TEE is delivered through a tarball file named **en.SOURCES-optee-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 OP-TEE

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [5]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package OP-TEE - STM32MP15-Ecosystem-v2.0.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-optee-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz
Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means: `$ cd <working directory path>/Developer-Package` • Download the tarball file in this directory • Uncompress the tarball file to get the OP-TEE (OP-TEE source code, ST patches...):

	STM32MP1 Developer Package OP-TEE - STM32MP15-Ecosystem-v2.0.0 release
	PC $> $ tar xvf en.SOURCES-optee-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz **PC $>** $ cd stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24/sources/arm-ostl-linux-gnueabi/optee-os-stm32mp-3.9.0.r1-r0 **PC $>** $ tar xvf optee-os-stm32mp-3.9.0.r1-r0.tar.gz
Release note	Details about the content of the OP-TEE are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **OP-TEE installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-20-06-24/sources/arm-ostl-linux-gnueabi* directory, and is named *optee-os-stm32mp-<OP-TEE version>*:

```
optee-os-stm32mp-3.9.0.r1-r0
├── [*].patch
├── optee-os-stm32mp-3.9.0.r1
├── Makefile.sdk
├── optee-os-stm32mp-3.9.0.r1-r0.tar.gz
├── README.HOW_TO.txt
└── series
```

```
OP-TEE installation directory
ST patches to apply during the OP-TEE preparation (see next chapter)
OP-TEE source code directory
Makefile for the OP-TEE compilation
Tarball file of the OP-TEE source code
Helper file for OP-TEE management: reference for OP-TEE build
List of all ST patches to apply
```

5.5.2 Building and deploying the OP-TEE for the first time

It is mandatory to execute once the steps specified below before modifying the OP-TEE.

As explained in the boot chains overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the OP-TEE is the *fsbl* one.

The README.HOW_TO.txt helper file is **THE** reference for the OP-TEE build

The SDK must be started

Open the *<OP-TEE installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the OP-TEE (optional but recommended)

prepare the OP-TEE (applying the ST patches)

cross-compile the OP-TEE

deploy the OP-TEE (i.e. update the software on board)

The OP-TEE is now installed: let's modify the OP-TEE.

5.6 Installing the debug symbol files

Optional step: **it is mandatory only if you want to debug Linux® kernel, U-Boot or TF-A with GDB.**

5.6.1 Downloading the debug symbol files

- The STM32MP1 debug symbol files is delivered through a tarball file named **en.DEBUG-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 debug symbol files

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [6]. If, trying to download, you encounter a "403 error", you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package debug symbol files - STM32MP15-Ecosystem-v2.0.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.DEBUG-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the debug symbol files (for Linux kernel, U-Boot, TF-A and OP-TEE OS):
	`PC $> $ tar xvf en.DEBUG-stm32mp1-openstlinux-5-4-dunfell-mp1-20-06-24.tar.xz`

	STM32MP1 Developer Package debug symbol files - STM32MP15-Ecosystem-v2.0.0 release
Release note	If you are interested in older releases, please have a look into the section Archives.

- The debug symbol files are in the *<Developer Package installation directory>/stm32mp1-openstlinux-20-06-24 /images/stm32mp1* directory:

```
stm32mp1
├── arm-trusted-firmware
│   ├── tf-a-bl2-optee.elf                    Debug symbol file for TF-A → TF-A for OP-TEE boot stage
│   ├── tf-a-bl2-serialboot.elf               Debug symbol file for TF-A → fsbl for flasher
│   ├── tf-a-bl2-trusted.elf                  Debug symbol file for TF-A → TF-A for OP-TEE boot stage
│   ├── tf-a-bl32-serialboot.elf              Debug symbol file for TF-A → secure monitor for flasher
│   └── tf-a-bl32-trusted.elf                 Debug symbol file for TF-A → runtime software stage
├── bootloader
│   ├── u-boot-stm32mp157a-dk1-optee.elf      Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157a-dk1-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157a-ev1-optee.elf      Debug symbol file for U-Boot → STM32MP15 Evaluation boards
│   ├── u-boot-stm32mp157a-ev1-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Evaluation boards
│   ├── u-boot-stm32mp157c-dk2-optee.elf      Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157c-dk2-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157c-ev1-optee.elf      Debug symbol file for U-Boot → STM32MP15 Evaluation boards
│   ├── u-boot-stm32mp157c-ev1-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Evaluation boards
│   ├── u-boot-stm32mp157d-dk1-optee.elf      Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157d-dk1-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157d-ev1-optee.elf      Debug symbol file for U-Boot → STM32MP15 Evaluation boards
│   ├── u-boot-stm32mp157d-ev1-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Evaluation boards
│   ├── u-boot-stm32mp157f-dk2-optee.elf      Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157f-dk2-trusted.elf    Debug symbol file for U-Boot → STM32MP15 Discovery kits
│   ├── u-boot-stm32mp157f-ev1-optee.         Debug symbol file for U-Boot → STM32MP15 Evaluation boards
```

```
elf
│   └── u-boot-stm32mp157f-ev1-
trusted.elf
├── kernel
│   └── vmlinux
└── optee
    ├── tee-stm32mp157a-dk1-optee.elf
    ├── tee-stm32mp157a-ev1-optee.elf
    ├── tee-stm32mp157c-dk2-optee.elf
    ├── tee-stm32mp157c-ev1-optee.elf
    ├── tee-stm32mp157d-dk1-optee.elf
    ├── tee-stm32mp157d-ev1-optee.elf
    ├── tee-stm32mp157f-dk2-optee.elf
    └── tee-stm32mp157f-ev1-optee.elf
```

```
Debug symbol file for Linux
kernel


Debug symbol file for OP-TEE O
S → STM32MP15 Discovery kits
Debug symbol file for OP-TEE O
S → STM32MP15 Evaluation
boards
Debug symbol file for OP-TEE O
S → STM32MP15 Discovery kits
Debug symbol file for OP-TEE O
S → STM32MP15 Evaluation
boards
Debug symbol file for OP-TEE O
S → STM32MP15 Discovery kits
Debug symbol file for OP-TEE O
S → STM32MP15 Evaluation
boards
Debug symbol file for OP-TEE O
S → STM32MP15 Discovery kits
Debug symbol file for OP-TEE O
S → STM32MP15 Evaluation
boards
```

5.6.2 Using the debug symbol files

These files are used to debug the Linux® kernel, U-Boot or TF-A with GDB. Especially, the Debug OpenSTLinux BSP components chapter explains how to load the debug symbol files in GDB.

6 Installing the components to develop software running on Arm Cortex-M4 (STM32Cube MPU Package)

6.1 Installing the Eclipse IDE

Optional step: it is needed if you want to modify or add software running on Arm Cortex-M.

The table below explains how to download and install the System Workbench for STM32 IDE which is the AC6 product addressing STM32 MCU, and the STM32-CoPro-MPU plugin which provides support for Cortex-M inside STM32 MPU.
STM32-CoPro-MPU plugin release note

6.2 Installing the STM32Cube MPU Package

Optional step: it is mandatory only if you want to modify the STM32Cube MPU Package.

Prerequite: the Eclipse IDE is installed.

- The STM32CubeMP1 Package is delivered through an archive file named **en.stm32cubemp1_v1-2-0.zip**.
- Download and install the STM32CubeMP1 Package

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

To download a package, it is recommended to be logged in to your "myst" account [7]. If, trying to download, you encounter a "403 error", you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

.

	STM32MP1 Developer Package STM32CubeMP1 Package - v2.0.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.stm32cubemp1_v1-2-0.zip

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:
	`$ cd <working directory path>/Developer-Package`
	• Download the archive file in this directory
	• Uncompress the archive file to get the STM32CubeMP1 Package:
	`$ unzip en.stm32cubemp1_v1-2-0.zip`

Release note	Details about the content of the STM32CubeMP1 Package are available in the *STM32Cube_FW_MP1_V2.0.0/Release_Notes.html* file. If you are interested in older releases, please have a look into the section Archives.

- The **STM32CubeMP1 Package installation directory** is in the *<Developer Package installation directory>* directory, and is named *STM32Cube_FW_MP1_V1.2.0*:

```
STM32Cube_FW_MP1_V1.2.0                          STM32CubeMP1 Package: details in ST
M32CubeMP1 Package content article
├── Drivers
│   ├── BSP                                      BSP drivers for the supported
```

```
STM32MP1 boards
│   │   └── [...]
│   ├── CMSIS
│   │   └── [...]
│   └── STM32MP1xx_HAL_Driver                    HAL drivers for the supported
STM32MP1 devices
│       └── [...]
├── _htmresc
│   └── [...]
├── License.md
├── Middlewares
│   └── [...]
├── package.xml
├── Projects
│   ├── STM32CubeProjectsList.html               List of examples and applications
for STM32CubeMP1 Package
│   ├── STM32MP157C-DK2                          Set of examples and applications →
STM32MP15 Discovery kits
│   │   └── [...]
│   └── STM32MP157C-EV1                          Set of examples and applications →
STM32MP15 Evaluation boards
│       └── [...]
├── Readme.md
├── Release_Notes.html                           Release note for STM32CubeMP1
Package
└── Utilities
    └── [...]
```

The STM32Cube MPU Package is now installed: let's develop software running on Arm Cortex-M4.

7 Developing software running on Arm Cortex-A7

7.1 Modifying the Linux kernel

Prerequisites:

- the SDK is installed
- the SDK is started up
- the Linux kernel is installed

The *<Linux kernel installation directory>/README.HOW_TO.txt* helper file gives the commands to:

configure the Linux kernel

cross-compile the Linux kernel

deploy the Linux kernel (that is, update the software on board)

You can refer to the following simple examples:

- Modification of the kernel configuration
- Modification of the device tree
- Modification of a built-in device driver
- Modification of an external in-tree module

7.2 Adding external out-of-tree Linux kernel modules

Prerequisites:

- the SDK is installed
- the SDK is started up
- the Linux kernel is installed

Most device drivers (or modules) in the Linux kernel can be compiled either into the kernel itself (built-in, or internal module) or as Loadable Kernel Modules (LKMs, or external modules) that need to be placed in the root file system under the /lib /modules directory. An external module can be in-tree (in the kernel tree structure), or out-of-tree (outside the kernel tree structure).

External Linux kernel modules are compiled taking reference to a Linux kernel source tree and a Linux kernel configuration file (*.config*).
Thus, a makefile for an external Linux kernel module points to the Linux kernel directory that contains the source code and the configuration file, with the **"-C <Linux kernel path>"** option.
This makefile also points to the directory that contains the source file(s) of the Linux kernel module to compile, with the **"M=<Linux kernel module path>"** option.

A generic makefile for an external out-of-tree Linux kernel module looks like the following:

```
# Makefile for external out-of-tree Linux kernel module

# Object file(s) to be built
obj-m := <module source file(s)>.o

# Path to the directory that contains the Linux kernel source code
# and the configuration file (.config)
KERNEL_DIR ?= <Linux kernel path>

# Path to the directory that contains the generated objects
DESTDIR ?= <Linux kernel installation directory>

# Path to the directory that contains the source file(s) to compile
PWD := $(shell pwd)

default:
	$(MAKE) -C $(KERNEL_DIR) M=$(PWD) modules

install:
	$(MAKE) -C $(KERNEL_DIR) M=$(PWD) INSTALL_MOD_PATH=$(DESTDIR) modules_install

clean:
	$(MAKE) -C $(KERNEL_DIR) M=$(PWD) clean
```

Such module is then cross-compiled with the following commands:

```
$ make clean
$ make
$ make install
```

You can refer to the following simple example:

- Addition of an external out-of-tree module

7.3 Adding Linux user space applications

Prerequisites:

- the SDK is installed
- the SDK is started up

Once a suitable cross-toolchain (OpenSTLinux SDK) is installed, it is easy to develop a project outside of the OpenEmbedded build system.
There are different ways to use the SDK toolchain directly, among which Makefile and Autotools.
Whatever the method, it relies on:

- the sysroot that is associated with the cross-toolchain, and that contains the header files and libraries needed for generating binaries (see target sysroot)
- the environment variables created by the SDK environment setup script (see SDK startup)

You can refer to the following simple example:

- Addition of a "hello world" user space application

7.4 Modifying the U-Boot

Prerequisites:

- the SDK is installed
- the SDK is started up
- the U-Boot is installed

The *<U-Boot installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the U-Boot

deploy the U-Boot (that is, update the software on board)

You can refer to the following simple example:

- Modification of the U-Boot

7.5 Modifying the TF-A

Prerequisites:

- the SDK is installed
- the SDK is started up
- the TF-A is installed

The *<TF-A installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the TF-A

deploy the TF-A (that is, update the software on board)

You can refer to the following simple example:

- Modification of the TF-A

7.6 Modifying the OP-TEE

Prerequisites:

- the SDK is installed
- the SDK is started up
- the OP-TEE is installed

The *<OP-TEE installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the OP-TEE

deploy the OP-TEE (that is, update the software on board)

8 Developing software running on Arm Cortex-M4

8.1 How to create a Cube project from scratch or open/modify an existing one from STM32Cube MPU package

Please refer to STM32CubeMP1 Package article.

9 Fast links to essential commands

If you are already familiar with the Developer Package for the STM32MPU Embedded Software distribution, fast links to the essential commands are listed below.

With the links below, you will be redirected to other articles; use the *back* button of your browser to come back to these fast links

Link to the command
Starter Packages
Essential commands of the STM32MP15 Evaluation board Starter Package
Essential commands of the STM32MP15 Discovery kit Starter Package
SDK
Download and install the latest SDK

Link to the command
Start the SDK
Linux kernel
Download and install the latest Linux kernel
Helper file for the Linux kernel build, and update on board
U-Boot
Download and install the latest U-Boot
Helper file for the **U-Boot** build, and update on board
TF-A
Download and install the latest TF-A
Helper file for the **TF-A** build, and update on board
Linux user space
Simple user space application
STM32Cube MPU Package
Download and install the latest STM32CubeMP1 Package
Create or modify a Cube project

10 How to go further?

Now that your developments are ready, you might want to switch to the STM32MP1 Distribution Package, in order to create your own distribution and to generate your own SDK and image.

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Trusted Firmware for Arm Cortex-A

Open Portable Trusted Execution Environment

Microprocessor Unit

Board support package

Hardware Abstraction Layer

(Software)Integrated development/design/debugging environment

GNU dedugger, a portable debugger that runs on many Unix-like systems

Display Serial Interface (MIPI® Alliance standard)

High-Definition Multimedia Interface (HDMI standard)

debug and test protocol, named from the Joint Test Action Group that developed it

Operating System

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Cortex Microcontroller Software Interface Standard