

# i.MX Windows 10 IoT User's Guide

for NXP i.MX Platform

Document Number: IMXWGU

Rev. W0.9.1, 3/2022



# Contents

<b>1</b>	<b>Overview</b>	<b>5</b>
1.1	Audience . . . . .	5
1.2	Conventions . . . . .	5
1.3	How to start . . . . .	5
1.4	References . . . . .	6
1.5	Feature List per Board . . . . .	6
<b>2</b>	<b>Deploy Windows IoT image to development board</b>	<b>8</b>
2.1	Hardware prerequisites . . . . .	8
2.2	Software prerequisites . . . . .	8
2.3	Basic Terminal Setup . . . . .	8
2.4	Basic Board Setup . . . . .	9
2.5	Install Windows IOT Enterprise to eMMC . . . . .	9
2.5.1	Preparing WinPE based installation media on a microSD card . . . . .	10
2.5.1.1	Prepare installation with prebuilt drivers . . . . .	10
2.5.1.2	Prepare installation building drivers from sources yourself . . . . .	10
2.5.1.3	Common steps . . . . .	10
2.6	Deploy boot firmware . . . . .	11
2.6.1	Firmware on eMMC . . . . .	11
2.6.2	Firmware on SD card . . . . .	12
<b>3</b>	<b>Building Windows 10 IoT for NXP i.MX Processors</b>	<b>13</b>
3.1	Obtaining BSP sources . . . . .	13
3.1.1	Required Tools . . . . .	13
3.1.2	The NXP i.MX Windows IoT BSP source package . . . . .	13
3.2	Building the drivers in the BSP . . . . .	15
3.2.1	Required Tools . . . . .	15
3.2.1.1	Visual Studio 2017 . . . . .	15
3.2.1.2	Windows Kits from Windows 10, version 1809 . . . . .	15
3.2.2	Structure of Windows driver sources . . . . .	16
3.2.3	One-Time Environment Setup . . . . .	16
3.2.4	Building the drivers . . . . .	16
3.2.5	How to use the signed prebuilt HAL drivers with the BSP . . . . .	17
3.3	Building ARM64 Firmware . . . . .	18
3.3.1	Setting up your build environment . . . . .	18

3.3.2	Building the firmware . . . . .	19
3.3.3	Common causes of build errors: . . . . .	20
<b>4</b>	<b>Revision History</b>	<b>21</b>

# 1 Overview

User's guide describes the process of building and installing the Windows 10 IoT OS BSP (Board Support Package) for the i.MX platform. It also covers special i.MX features and how to use them.

Guide also lists the steps to run the i.MX platform, including board DIP switch settings, and instructions on the usage and configuration of U-Boot boot loader.

Features covered in this guide may be specific to particular boards or SOCs. For the capabilities of a particular board or SOC, see the *i.MX Windows 10 IoT Release Notes* (IMXWIN10RN).

## 1.1 Audience

This chapter is intended for software, hardware, and system engineers who are planning to use the product, and for anyone who wants to know more about the product.

## 1.2 Conventions

This chapter uses the following conventions:

- Courier New font: This font is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.

## 1.3 How to start

The i.MX Windows 10 IoT BSP is a collection of binary files, source code, and support files you can use to create a bootable Windows 10 IoT image for i.MX development systems.

Before you start, see the [Feature List per Board chapter](#). This section lists all the i.MX boards covered by this BSP and also contains a list of possible features.

If you have downloaded i.MX Windows 10 IoT binary BSP package, please go to Deploy Windows IoT image to development board to create flash bootloader and create SD card installer.

If you have downloaded an archive with BSP sources, please go to Building Windows 10 IoT for NXP i.MX Processors ) and check the process of the building the Windows drivers and Boot firmware. After that you can prepare Windows installer on SD card and flash bootloader according to Deploy Windows IoT image to development board chapter.

## 1.4 References

For more information about Windows 10 IoT Enterprise, see Microsoft online documentation.

- <http://windowsondevices.com>

The quick start guides contain basic information on the board and setting it up. They are on the NXP website.

- [i.MX 8M Quad Evaluation Kit Quick Start Guide \(IMX8MQADEVKQSG\)](#)

Documentation is available online at [nxp.com](http://nxp.com)

- i.MX 8 information is at <http://www.nxp.com/imx8>

## 1.5 Feature List per Board

Table 1.1: Overview of the currently supported features for every board.

Feature	MCIMX8M-EVK
BSP name	NXPEVK_IMX8M_4GB
SD Card boot	Y
eMMC boot	Y
Audio	Y
GPIO	Y
I2C	Y
Ethernet	Y
PWM	Y
SD Card	Y
eMMC	Y
SPI (master)	N/A
Display	Y
UART	Y*

Feature	MCIMX8M-EVK
USB (host)	Y
PCIe	Y
TrEE	Y
M4	N**
Processor PM	Y
Device PM	N**
LP standby	N**
Display PM	Y
DMA	Y

Legend	Meaning
Y	Enabled
Y*	To enable the UART, the kernel debugger must be disabled by running the following command on the device and rebooting. The UART exposed is the same UART that the kernel debugger uses. <code>bcdedit /debug off</code>
N/A	Feature not applicable
N*	Feature not enabled - feature is not available in default board configuration
N**	Feature not enabled - feature is not currently supported
PM	Power management
LP	Low power

Not all features of a given subsystem maybe fully enabled and/or optimized. If you encounter issues with supported features, please open an issue.

## 2 Deploy Windows IoT image to development board

This chapter will guide you through the deployment process of Windows operating system and boot firmware to the development board.

### 2.1 Hardware prerequisites

- NXP i.MX development board (additional information can be found in Supported boards table in i.MX\_Win10\_Release\_Notes.pdf)
- USB-C cable to connect from device to PC
- microSD card (minimum 8GB)

### 2.2 Software prerequisites

- Binary drivers (either downloaded from NXP.com or built locally)
- Windows IoT Enterprise version 2004 ARM64 OS image. <https://partner.microsoft.com/en-us/dashboard/collaborate/packages/12392>
- Windows ADK 1809. If you are not building firmware on the same PC, you can install Windows ADK for Windows 10, version 2004. <https://docs.microsoft.com/en-us/windows-hardware/get-started/adk-install>
- Windows PE add-on for the ADK, version 2004. Regardless of whether you installed ADK version 1809 or 2004, you should install version 2004 of WinPE. <https://docs.microsoft.com/en-us/windows-hardware/get-started/adk-install>

### 2.3 Basic Terminal Setup

During the boot, you can check the U-Boot and UEFI firmware output on the host PC by using the serial interface console. In the case you don't see any output on the connected display, for example, this might be helpful to confirm that the board is booting. Common serial communication applications such as HyperTerminal, Tera Term, or PuTTY can be used on the host PC side. The i.MX boards connect the host driver using the micro-B USB connector.

1. Connect the target and the PC using a cable mentioned above.



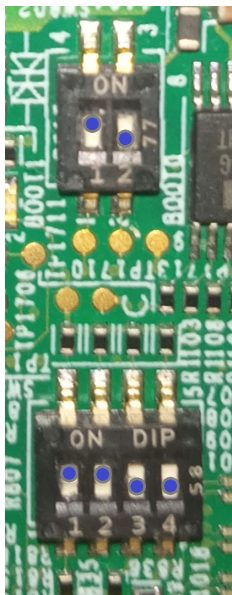
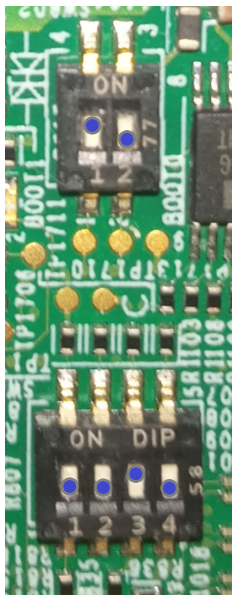
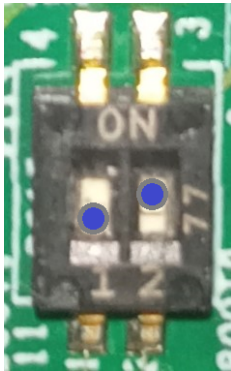
2. Open terminal on the PC. Configure serial to 921600 baud/s, 8 bit, one stop bit.

You may need to download drivers for USB to serial converter. The USB to serial driver for FTDI chip can be found under <http://www.ftdichip.com/Drivers/VCP.htm>. The FTDI USB to serial converters provide up to four serial ports. The USB to serial VCP (Virtual COM Port) driver for CP210x chip can be found under <https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>. The CP210x USB to serial converters provide up to two serial ports.

The order of serial ports differ between boards sometimes even Windows hosts differ in COM port order. For MCIMX8M-EVK board user need to select the first port (COM) in terminal to read bootloader messages or debug kernel with WinDbg.

## 2.4 Basic Board Setup

Table 2.1: MCIMX8M-EVK boot DIP cfg

SD card boot	eMMC boot	USB UUU Download
		

## 2.5 Install Windows IOT Enterprise to eMMC

Since eMMC is soldered to the board, the only way to write to it is to boot manufacturing OS on the device, then write the OS image to eMMC. The manufacturing OS is booted from a removable storage

such as USB or SD. In this document, we will list commands for an MCIMX8M-EVK board to show an example of an ARM64 board deployment. The tools and techniques can be adapted to other hardware designs.

For the manufacturing OS, we will use [Windows PE \(WinPE\)](#), which stands for Windows Preinstallation Environment. Windows PE is a small Windows image that can boot without persistent storage, and contains tools to help install Windows such as [diskpart](#) and [dism](#).

The high-level process we will follow is:

1. Prepare a microSD card with the OS image.
2. Install the prepared OS image to eMMC through microSD card.

### 2.5.1 Preparing WinPE based installation media on a microSD card

#### 2.5.1.1 Prepare installation with prebuilt drivers

1. Download W21H2-X.X.X-imx-windows-bsp-binaries.zip and extract it.
2. Open elevated command prompt and navigate to extracted `<imx-windows-bsp-binaries>\IoTEntOnNXP`.
3. Continue to [Common steps](#)

#### 2.5.1.2 Prepare installation building drivers from sources yourself

1. Build the drivers yourself as described in [Building Windows 10 IoT for NXP i.MX Processors](#).
2. After the BSP has been built open elevated command prompt and navigate to `<BSP_DIR>\imx-windows-iot\BSP\IoTEntOnNXP`.
3. Continue to [Common steps](#)

#### 2.5.1.3 Common steps

1. Download the Windows IoT Enterprise version ISO mentioned in [Software prerequisites](#).
2. After the ISO file have been downloaded, mount the ISO file and copy out the install.wim from `<DVD mount drive:>\sources\install.wim` to the `IoTEntOnNXP` directory.
3. In the `IoTEntOnNXP` directory run `make-winpe-enterprise.cmd` in elevated prompt. This will create copy of selected install.wim image with injected i.MX drivers. Use `/disable_updates` to save storage space and `/disable_transparency` to improve UI responsiveness. Use `/patch_sdport` to update Sdport.sys. This option should to enabled in case you are creating installer for the

W10-19044.1288.211006 Windows build. Certain operations with Sdport.sys cause the blue bug check screen. The `/image <WIM_PATH>` could be either absolute or relative to `IoTEntOnNXP\work`. Work might not exist yet. The script will create it and then enter it.

```
1 make-winpe-enterprise.cmd /device NXPEVK_iMX8M_4GB /disable_updates /disable_transparency /
  patch_sdport /image ..\install.wim
```

4. Now you can deploy the WinPE image to an SD card by `make-winpe-enterprise.cmd /apply <disk_number>`. The `disk_number` of your SD card can be obtained using `diskpart` or using `wmic diskdrive list brief` from the DeviceID string (`\\.\PHYSICALDRIVE<disk_number>`).

```
1 make-winpe-enterprise.cmd /device NXPEVK_iMX8M_4GB /apply <disk_number>
```

5. WinPE based Windows installer is now deployed on the SD card.

Installer also renames the `EFI` folder at the root of SD card to `_efi`, which causes UEFI to skip the SD card at boot time. Bootloader will find the `EFI` directory only on eMMC, and boot from there. If you wish to boot into the WinPE installer again, you can rename `_efi` back to `EFI`.

6. After you deploy boot firmware (see [Deploy boot firmware](#)) you can insert the prepared SD card installer into your development board and power it on. Board will boot into the installer. Installer will flash Windows IOT Enterprise to the eMMC, then reboot into installed Windows.

## 2.6 Deploy boot firmware

There is prebuilt boot firmware (`firmware.bin`) included in both binary and source BSP packages. This boot firmware could be deployed to both SD card and onboard eMMC. Boot source is determined by DIP switches.

### 2.6.1 Firmware on eMMC

For eMMC deployment we recommend to use UUU (Universal Update Utility tool for NXP IMX chips). The UUU tool requires U-boot with UUU download support enabled. Thus the BSP ships with UUU enabled U-Boot that is not recommended for final designs. To flash boot firmware to the eMMC we recommend to download UUU tool 1.3.191 which was used for BSP testing from [https://github.com/NXPmicro/mfgtools/releases/tag/uuu\\_1.3.191](https://github.com/NXPmicro/mfgtools/releases/tag/uuu_1.3.191).

To deploy boot firmware to eMMC via USB download mode follow steps below:

1. Navigate to `firmware` directory. In the source package it is located in `imx-windows-iot/BSP/firmware`.
2. Download UUU tool and put it into `firmware\tools\uuu\uuu.exe`.

3. Begin preparing board by powering it off.
4. Switch the target board to USB download mode using DIP switches.
5. Connect a USB-C cable between your desktop and the board's USB-C port.
6. Power on the board and run `flash_bootloader.cmd /device NXPEVK_iMX8M_4GB` inside `firmware` directory.
7. When the process finishes power the board off. Disconnect the USB-C cable connecting board to PC. Not disconnecting from PC may cause errors in later steps.
8. Switch the board to eMMC boot mode. See [Basic Board Setup](#)

UUU documentation can be found at <https://github.com/NXPmicro/mfgtools>.

### 2.6.2 Firmware on SD card

Firmware on SD card is recommended for development to avoid dip switch manipulation. The boot firmware could be deployed to SD card containing WinPE based installer as described in [Install Windows IOT Enterprise to eMMC](#). However WinPE instaler wipes SD card. Thus WinPe must be deployed first. To deploy boot firmware to SD card from Windows we recommend to use Cfimager tool from [https://www.nxp.com/webapp/Download?colCode=CF\\_IMAGER](https://www.nxp.com/webapp/Download?colCode=CF_IMAGER).

1. Navigate to `firmware` directory. In the source package it is located in `imx-windows-iot/BSP/firmware`.
2. Download NXP cfimager tool and put it into `firmware/cfimager.exe` or system `%PATH%`.
3. Power on the board and run `flash_bootloader.cmd /device NXPEVK_iMX8M_4GB /target_drive <SD card letter eg. f:>` inside `firmware` directory.
4. Begin preparing board by powering it off.
5. Switch the target board to boot form SD card using DIP switches. See [Basic Board Setup](#)

## 3 Building Windows 10 IoT for NXP i.MX Processors

### 3.1 Obtaining BSP sources

#### 3.1.1 Required Tools

The following tools are required to follow this chapter:

- git
- git-lfs
- software to unpack zip, gzip and tar archives

#### 3.1.2 The NXP i.MX Windows IoT BSP source package

The Windows IoT BSP for NXP i.MX Processors consists of multiple parts, namely:

- 1) The NXP i.MX BSP sources package which is available at [www.NXP.com](http://www.NXP.com). Package contains sources for both the boot firmware and Windows drivers.
- 2) The iMX firmware and NXP Code Signing Tool (CST) available at [www.NXP.com](http://www.NXP.com).
- 3) Open-source parts of boot firmware and tools that are not distributed as part NXP BSP package.
- 4) The Windows IOT Enterprise operating system provided by Microsoft.

To prepare sources for building BSP follow these steps:

Note Only steps 1-3 are required for building Windows drivers from source. All steps are required for building firmware from source.

- 1) Download the NXP i.MX BSP provided as `W<os_version>-imx-windows-bsp-<build_date>.zip` archive from [www.NXP.com](http://www.NXP.com).
- 2) Create an empty directory, further referred as `<BSP_DIR>`, and extract the downloaded archive there. This will create .git repository containing the BSP release. Shell command `unzip W<os_version>-imx-windows-bsp-<build_date>.zip -d win10-iot-bsp` command can be used to extract the package.
- 3) Populate the directory by running `Init.bat` or `Init.sh` as applicable to the platform. Both scripts will checkout sources from the repository by `git reset --hard`. The `Init.sh` shall also

checkout submodules that are required to build i.MX boot firmware by `git submodule update --init --recursive`. During prerelease testing the `Init.sh` executed inside Ubuntu environment run into “server certificate verification failed. CAfile: /etc/ssl/certs/ca-certificates.crt CRLfile: none” error. Problem could be solved by installing `apt-transport-https ca-certificates` and certificate update.

```
1 sudo apt update ; sudo apt-get install apt-transport-https ca-certificates -y ; sudo
   update-ca-certificates
```

- 4) At this point it is possible to build the Windows drivers. Further steps are required only to build i.MX boot firmware (ATF, U-boot and UEFI) from sources.
- 5) Download and extract the [Code Signing Tools \(CST\)](#) from NXP’s website. You will need to create an account on NXP’s website to access this tool. Extract the tool inside bsp repository, and rename the newly created folder to `cst` to get `<BSP_DIR>/cst` folder:

```
1 tar xf cst-3.1.0.tgz
2 mv release cst
3 rm cst-3.1.0.tgz
```

- 6) Download and extract the [iMX firmware](#) from NXP’s website and place it to `firmware-imx`. Extract the tool inside bsp repository, and rename the newly created folder to `firmware-imx` to get `<BSP_DIR>/firmware-imx/firmware/ddr/` in directory tree:

```
1 chmod +x firmware-imx-8.3.bin
2 ./firmware-imx-8.3.bin
3 mv firmware-imx-8.3 firmware-imx
4 rm firmware-imx-8.3.bin
```

- 7) At this point your directory structure should contain the following folders.

```
1 - <BSP_DIR>
2   |- cst (manually downloaded)
3   |- firmware-imx (manually downloaded)
4   |- Documentation
5   |- MSRSec
6   |- RIOT
7   |- imx-atf
8   |- imx-mkimage
9   |- imx-optee-os
10  |- imx-windows-iot
11  |- mu_platform_nxp
12  |- patches
13  |- uboot-imx
```

## 3.2 Building the drivers in the BSP

Before you start building the BSP, you need to have an archive with latest BSP sources from NXP sites downloaded and extracted as described in [Obtaining BSP sources](#) chapter.

### 3.2.1 Required Tools

The following tools are required to build the drivers:

- Visual Studio 2017
- Windows Kits (ADK/SDK/WDK)

#### 3.2.1.1 Visual Studio 2017

- Make sure that you **install Visual Studio 2017 before the WDK** so that the WDK can install a required plugin.
- Download [Visual Studio 2017](#).
- During install select **Desktop development with C++**.
- During install select the following in the Individual components tab. If these options are not available try updating VS2017 to the latest release:
  - **VC++ 2017 version 15.9 v14.16 Libs for Spectre (ARM)**
  - **VC++ 2017 version 15.9 v14.16 Libs for Spectre (ARM64)**
  - **VC++ 2017 version 15.9 v14.16 Libs for Spectre (X86 and x64)**
  - **Visual C++ compilers and libraries for ARM**
  - **Visual C++ compilers and libraries for ARM64**

#### 3.2.1.2 Windows Kits from Windows 10, version 1809

- **IMPORTANT: Make sure that any previous versions of the ADK and WDK have been uninstalled!**
- Install [ADK 1809](#)
- Install [WDK 1809](#)
  - Scroll down and select Windows 10, version 1809.
  - Make sure that you allow the Visual Studio Extension to install after the WDK install completes.

- If the WDK installer says it could not find the correct SDK version, install [SDK 1809](#)
  - Scroll down and select Windows 10 SDK, version 1809 (10.0.17763.0).
- After installing all Windows Kits, restart computer and check if you have correct versions installed in Control panel.

### 3.2.2 Structure of Windows driver sources

The *imx-windows-iot* - sources of Windows drivers has the following structure:

BSP - Contains boot firmware, driver binaries (Generated at build time.) and scripts needed to deploy BSP to development board.

build - Contains build scripts, and the VS2017 solution file.

components - Contains third party binaries and utility projects.

driver - Contains driver sources.

hals - Contains hal extension sources.

### 3.2.3 One-Time Environment Setup

Test certificates must be installed to generate driver packages on a development machine.

1. Open an Administrator Command Prompt.
2. Navigate to your BSP and into the folder `imx-windows-iot\build\tools`.
3. Launch `StartBuildEnv.bat`.
4. Run `SetupCertificate.bat` to install the test certificates.

Some tools may not work correctly if LongPath is not enabled, therefore run following command in console:

1. Execute `reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem /v LongPathsEnabled /t REG_DWORD /d 1` command.

### 3.2.4 Building the drivers

1. Open the solution `imx-windows-iot\build\solution\iMXPlatform\iMXPlatform.sln` located in the path where you have extracted BSP archive.



Note Microsoft pkggen requires launching Visual Studio 2017 as Administrator. Although driver projects are configured not to generate packages BSP might contain miscellaneous projects that require launching Visual Studio as Administrator to build successfully.

2. Choose [Debug](#) or [Release](#) build type.
3. If secure boot feature is enabled it is required to use signed drivers. For details see [How to use the signed prebuilt HAL drivers with the BSP](#) section.
4. To build press Ctrl-Shift-B or choose Build -> Build Solution from menu. This will compile all driver packages then `imx-windows-iot\BSP\IoTEntOnNXP\drivers` for deployment.
5. The updated drivers could now be injected into the installation image or manually installed to running development board.
  - To manually install drivers it has to be copied to development board via USB drive, network share, scp, remote desktop. Driver can be installed either by clicking [install](#) in right click menu of 'inf' file or by devcon command-line utility.
  - For debug use the [.kdfiles](#) of WinDBG. Then to initiate driver reload either use devcon or reset the board.
  - To create new installation SD card follow [Deploy Windows IoT image to development board](#)

### 3.2.5 How to use the signed prebuilt HAL drivers with the BSP

The Secure Boot feature of Windows 10 requires use of OEM signed drivers. Certificates are then validated with certification authority.

HAL drivers however has to be signed by Microsoft certification authority. HAL is therefore provided prebuilt and signed with the BSP sources inside `imx-windows-iot\components\NXP.SignedDrivers\BootDrivers.<ARCHITECTURE>` directory.

1. To use these prebuilt signed drivers in BSP project modify the `imx-windows-iot\build\tools\CreateBSP` to copy HAL drivers from `imx-windows-iot\components\NXP.SignedDrivers` instead of the build results.
2. After building the driver project as described in step 4 of [Building the drivers](#) the signed hal will be ready for deployment into the installer using `make-winpe-enterprise.cmd` (see [Deploy Windows IoT image to development board](#)).

## 3.3 Building ARM64 Firmware

This chapter describes the process of setting up a build-environment to build the latest firmware and update the firmware on development board.

### 3.3.1 Setting up your build environment

1. Start Linux environment such as:

- Dedicated Linux system
- Linux Virtual Machine
- Windows Subsystem for Linux ([WSL setup instructions](#))

Note: W-imx-windows-bsp-.zip was validated with Ubuntu 20.04 in WSL and also standalone Ubuntu.

2. Install or update build tools. On Ubuntu run the following commands from shell.

```
1  sudo apt-get update
2  sudo apt-get upgrade
3
4  # Ubuntu 18.04 and possibly other distributions might require newer version of mono than present in
   # official repository. This could be solved by adding mono repository to the system.
5  # Process for Ubuntu 18.04 taken from https://www.mono-project.com/download/stable/#download-lin:
6  # sudo apt install gnupg ca-certificates
7  # sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
   3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
8  # # Optionally key could be downloaded to a file and added manually by 'apt-key add <keyfile>'.
9  # # Now that certificate is installed we can add official mono repository to repository list.
10 # echo "deb https://download.mono-project.com/repo/ubuntu stable-bionic main" | sudo tee /etc/apt/
   sources.list.d/mono-official-stable.list
11 # sudo apt update
12
13 sudo apt-get install attr build-essential python3.8 python3.8-dev python3.8-venv
   device-tree-compiler bison flex swig iasl uuid-dev wget git bc libssl-dev zlib1g-dev
   python3-pip mono-devel gawk
14 # The mu_project currently requires python 3.8 and higher. Create a new python environment for the
   build and activate it.
15 python3.8 -m venv ~/venv/win_fw_build
16 source ~/venv/win_fw_build/bin/activate
17
18 # Get cross compiler
19 pushd ~
20 wget https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/aarch64-linux-gnu/
   gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz
21 tar xf gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz
22 rm gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz
23 # The cross compiler prefix is required to be exported later into AARCH64_TOOLCHAIN_PATH variable.
```

```
24 # export AARCH64_TOOLCHAIN_PATH=~/.gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin/
    aarch64-linux-gnu-
25 popd
26
27 # Following steps require to be done inside BSP_DIR extracted from
    W<os_version>-imx-windows-bsp-<build_date>.zip.
28 cd <BSP_DIR>
29 pushd mu_platform_nxp
30 pip3 install -r requirements.txt --upgrade
31
32 # The pycrypto package is obsolete. Remove it if installed
33 sudo apt remove python3-crypto # installed using apt
34 pip3 uninstall pycrypto # installed using pip3
35 # And install pycryptodome
36 pip3 install pycryptodome
37 python3 NXP/MCIMX8M_EVK_4GB/PlatformBuild.py --setup # If facing problems during UEFI build use
    --force to clean the environment. Commit or stash all your changes first, command performs git
    reset --hard.
38 python3 NXP/MCIMX8M_EVK_4GB/PlatformBuild.py --update # If command fails try reinstalling mono from
    its official repository as noted above.
39 popd
```

### 3.3.2 Building the firmware

This chapter assumes the BSP sources have been prepared as described in [Obtaining BSP sources](#) chapter.

To build the boot firmware export `AARCH64_TOOLCHAIN_PATH` and execute the `buildme64.sh -b <BOARD_NAME> -t all [-clean]` script provided in the root of BSP sources. The `buildme64.sh` script bundled in BSP also copies `flash.bin` and `uefi.fit` into `<BSP_DIR>/imx-windows-iot/component/firmware/<board_name>` to allow for partial (only UEFI or U-boot) rebuild.

In case of BSP update, you may need to run `python3 NXP/MCIMX8M_EVK_4GB/PlatformBuild.py --setup --force` inside `mu_platform_nxp` to get clean and up-to-date MU build environment. Make sure to stash or commit your changes. Command performs `git reset --hard`.

```
1 # export AARCH64_TOOLCHAIN_PATH pointing at the cross compiler
2 # eg. export AARCH64_TOOLCHAIN_PATH=~/.gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin/
    aarch64-linux-gnu-
3 # Get into virtual environment
4 source ~/venv/win_fw_build/bin/activate
5 ./buildme64.sh -b 8M -t all -c -fw
```

To deploy `firmware.bin` to the i.MX development board follow the process described in [Deploy boot firmware](#).

### 3.3.3 Common causes of build errors:

1. ImportError: No module named Crypto.PublicKey.
  - This error is encountered when pycryptodome module is missing. Optionally obsolete pycrypto needs to be removed.
2. Unable to enter directory. Directory doesn't exist.
  - We have run into this problem in case gitmodules were not downloaded completely (eg. MSRSec is empty) or `cst` or `firmware` directories are missing. Try repeating the [Obtaining BSP sources](#) step by step.
3. The build fails in WSL while the BSP is located somewhere in `/mnt/c` of the WSL.
  - Try `setfatattr -n system.wsl_case_sensitive -v 1 <BSP_DIR>`. OP-Tee also requires symbolic links. We have been able to build boot firmware in `/mnt/c/` on Windows os version 1909. Workaround is to copy the BSP to WSL filesystem eg. to HOME.
4. RuntimeError: SDE is not current. Please update your env before running this tool.
  - Try to update `mono_devel` package and repeat setup and update steps for PlatformBuild.py in [Setting up your build environment](#).
5. cp: cannot stat '././firmware-imx/firmware/ddr/synopsys/lpddr4\_pmu\_train\_\*.bin'
  - Please make sure the firmware-imx obtained in [Obtaining BSP sources](#) chapter is placed inside the BSP in way the following directories exist `<BSP_DIR>/firmware-imx/firmware/ddr/synopsys`.

## 4 Revision History

Table 4.1: Revision history

Revision number	Date	Substantive changes
W0.9.0	1/2022	Initial engineering release for i.MX8M platform.
W0.9.1	3/2022	Initial engineering release for i.MX8M platform bugs fixes.