# AN13739

## Visual Studio and Eclipse Debugging for i.MX 8M Family

**Rev. 0 — 11 November 2022** **Application note**

**Document information**

| Information | Content |
|---|---|
| Keywords | AN13739, i.MX 8M, Yocto, Eclipse IDE, Microsoft Visual Studio IDE |
| Abstract | This document provides multiple solutions for cross-compilation, deployment, and debugging of applications that target the i.MX 8M family. |

# 1 Introduction

This document describes the processes to set up the Windows software development environment for the i.MX 8M family devices in various development environments.

In addition, it helps the user to cross-compile, deploy, and debug the code for an i.MX 8M device in the following two Integrated Development Environments (IDEs):

- Microsoft Visual Studio
- Eclipse

The programming languages covered in this document are:

- C/C++
- Python

The main objectives of this document are:

- Describing the board setup for both hardware and software. The setup process includes the steps to build the image from scratch using the Yocto project.
- Setting up the connection from PC to board.
- Setting up IDEs for the programming languages, such as C and Python. The IDEs setup configurations include code and project examples.

# 2 Prerequisites

This section provides the required list of hardware and software components that should be installed in prior to setting up the Windows software development environment.

## 2.1 Hardware

Table 1 lists the required hardware components:

**Table 1. Hardware Components**

| Component | Description |
|---|---|
| NXP i.MX 8M | Mini EVK board |
| SanDisk Ultra 32 GB Micro SDHC | Micro SD card |
| USB Type-A to Type-C, or USB Type-C | Power cable for power port |
| Ethernet cable | Cable to connect the devices of the network |

## 2.2 Software

The configuration of Eclipse and Visual Studio environments require specific software tools.

Table 2 lists the required software components:

**Table 2. Software components**

| Environment | Software component | Description |
|---|---|---|
| Linux | Linux BSP Release 5.10.72_2.2.0 | Build Linux environment and customized with Yocto. |
| Eclipse | C/C++ | Develop applications in C or C++ using Eclipse IDE. |

**Table 2. Software components**...*continued*

| Environment | Software component | Description |
|---|---|---|
| | Download link for Eclipse: https://www.eclipse.org/downloads/packages/<br>Toolchain installer file for Windows: gcc-arm-10.3-2021.07-mingw-w64-i686-aarch64-none-linux-gnu.tar.xz | Develop applications in C or C++ using Eclipse IDE. |
| | Python for Java<br>Java version used in this context is 2022-03 R | Develop applications in Python using Eclipse IDE. |
| Visual Studio | Microsoft Visual Studio 2022 | Develop applications in C++ using Microsoft Visual Studio IDE. |

# 3 Yocto project

This section describes the Linux image customizations and Yocto file modifications that are required for the Yocto project setup.

The Yocto project is an open source collaboration project that helps developers create a customized Linux-based system regardless of the hardware architecture.

## 3.1 Customizing Linux image

Eclipse uses specific ciphers that are not available in the default SSH server manager on the Dropbear board. The following specific customizations of the image are required for connecting to the board through SSH:

- Changing the default SSH server manager on the board (Dropbear) to the server manager compatible with the presented IDEs (OpenSSH).
- Including the SSH File Transfer Protocol (SFTP) tools to the image. These SFTP tools are required for remote file transfer between the board and the presented IDEs.

The following list of tools is required for remote debugging:

- `/usr/libexec/sftp-server`
- `/usr/bin/sftp`
- `/usr/bin/gdbserver`
- `/usr/lib/libstdc++.so.6`

***Note:*** *The tools listed above are included in the Linux image for the build.*

## 3.2 Modifying Yocto file

To remove Dropbear from the board, the Yocto file `local.conf` parameters are modified. In addition, the OpenSSH package is customized for compilation.

To modify the Yocto recipe parameters, perform the following steps on a Linux host or VM:

1. Run the following commands:

```
$ mkdir ~/bin ; (This step is may not required, if the bin
 folder already exists.)
```

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo
 > ~/bin/repo
$ chmod a+x ~/bin/repo
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config --list
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://source.codeaurora.org/external/imx/
imx-manifest -b imx-linux-hardknott -m imx-5.10.72-2.2.0.xml
$ repo sync
$ DISTRO=fsl-imx-xwayland MACHINE=imx8mm-lpddr4-evk source
 imx-setup-release.sh -b build-comp
```

2. To remove Dropbear, perform the following steps:
   a. Open the recipe `imx-yocto-bsp/sources/meta-imx/meta-sdk/recipes-fsl/images/imx-image-core.bb`
   b. Locate the following lines in the code:

```
IMAGE_FEATURES += " \
    debug-tweaks \
    tools-profile \
    tools-sdk \
    package-management \
    splash \
 nfs-server \
 tools-debug \
 ssh-server-dropbear \
hwcodecs \
${@bb.utils.contains('DISTRO_FEATURES', 'wayland',
 'weston','', d)} \
"
```

3. Remove the line `ssh-server-dropbear \` from the yocto recipe `imx-yocto-bsp/sources/meta-imx/meta-sdk/recipes-fsl/images/imx-image-core.bb`.
4. Open the yocto recipe `imx-yocto-bsp/sources/meta-imx/meta-sdk/recipes-fsl/images/imx-image-multimedia.bb`
5. Remove the line `ssh-server-dropbear \` (same line as in the yocto recipe `imx-image-core.bb`).
6. Modify the line `imx-yocto-bsp/build-comp/conf/local.conf` by adding the following lines at the end of it:

```
#SSH
PACKAGE_EXCLUDE += " packagegroup-core-ssh-dropbear"
CORE_IMAGE_EXTRA_INSTALL += "openssh"
```

7. From the directory path `imx-yocto-bsp/build-comp`, build the image `image imx-image-multimedia` using the following command:

```
$ bitbake imx-image-multimedia
```

For more information regarding the Yocto image build, refer to *i.MX Yocto Project User's Guide* (document IMXLXYOCTOUG).
The built image name and its location are given below:

- **Location**: `imx-yocto-bsp/build-comp/tmp/deploy/images/imx8mmevk`

- **Image name**: `imx-image-multimedia-imx8mmevk.wic.bz2`.

8. After completion of the build, connect the board through Putty using the following steps:
   a. Flash the built image on the SD card.
   b. Insert the SD card in the board and then connect the board through Putty.
   For more information on image flashing and board connection process, refer to *i.MX Linux User's Guide* (document IMXLUG).

# 4 Configuring LAN

This section helps the user to create Local Area Network (LAN) between the host PC and the board using an Ethernet cable. In addition, it provides the steps to assign the static IP addresses to the interfaces on each end of the cable.

Figure 1 shows the Ethernet ports connecting the host PC and the board:

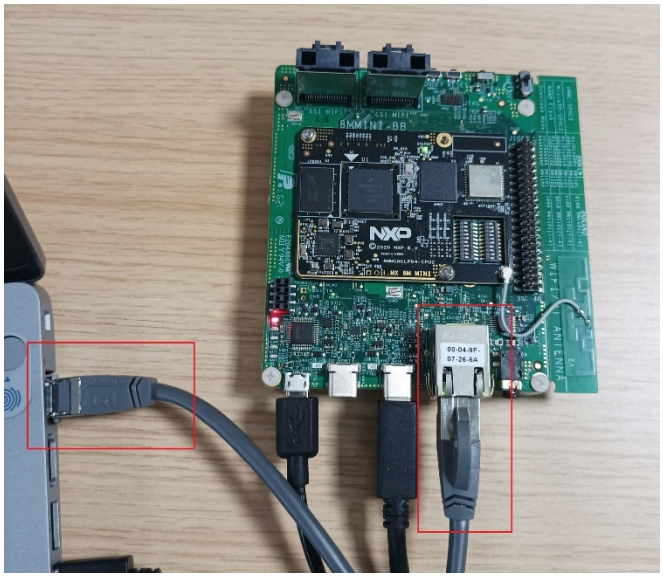

**Figure 1. Ethernet cable**

## 4.1 Host PC configuration

To configure the IP address for the host PC, perform the following steps:

1. On the host PC, navigate through **Control Panel > Network and Internet > Network and Sharing Center**.
2. Click **Network Status and Tasks** to view the network connections.
3. From **Connections**, click the connected network.
   For example, `Ethernet` is the connected network in the Figure 2.

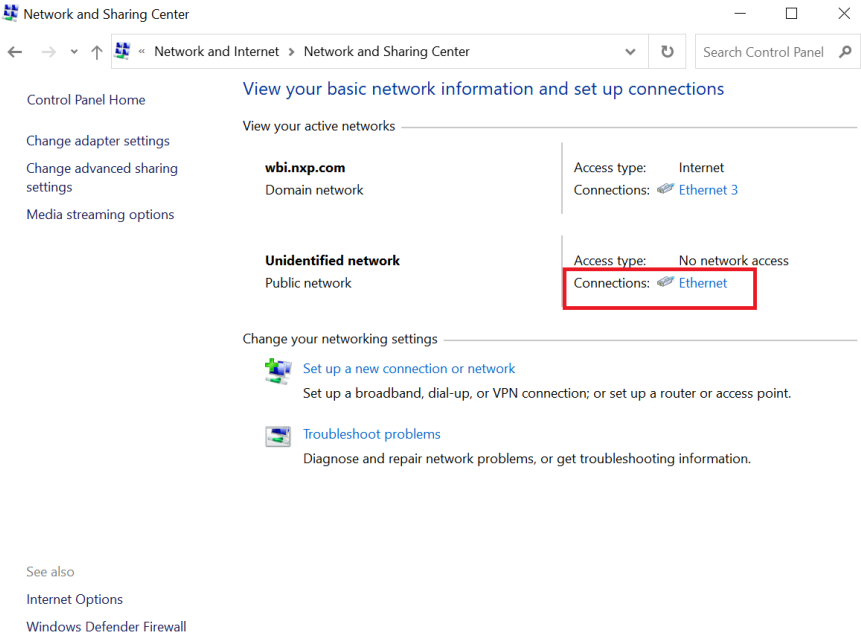


**Figure 2. Network and sharing center**

4. Click **Properties**. The **Ethernet Properties** dialog box appears.

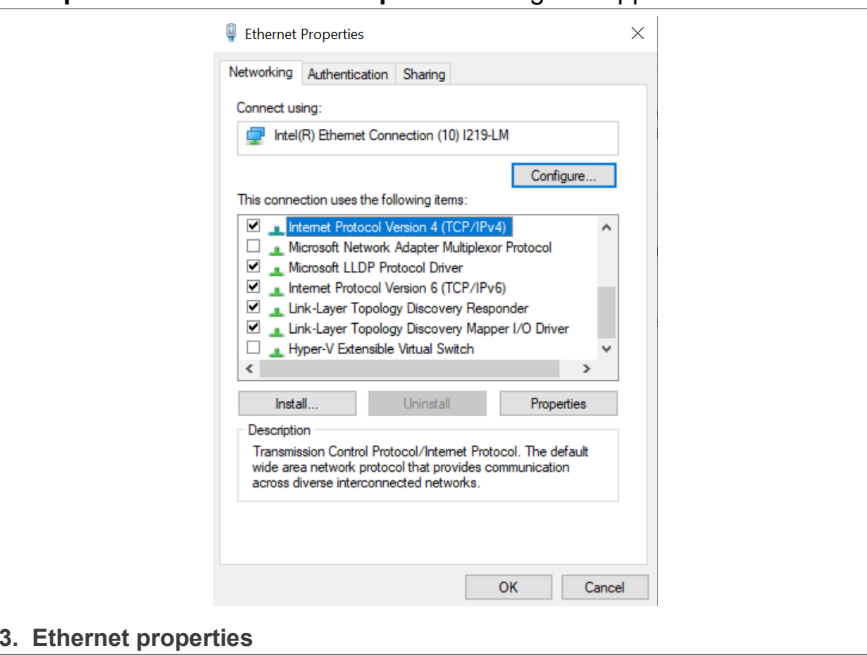


**Figure 3. Ethernet properties**

5. In the **Ethernet Properties** dialog box, select **Internet Protocol Version 4 (TCP/IPv4)** and click **OK**.
   The **Internet Protocol Version 4 (TCP/IPv4) Properties** dialog box appears.
6. In the **Internet Protocol Version 4 (TCP/IPv4) Properties** dialog box, configure the static IPv4 address for the interface assigned to the PC (from the Ethernet cable). Use the IP address details provided in the Figure 4.

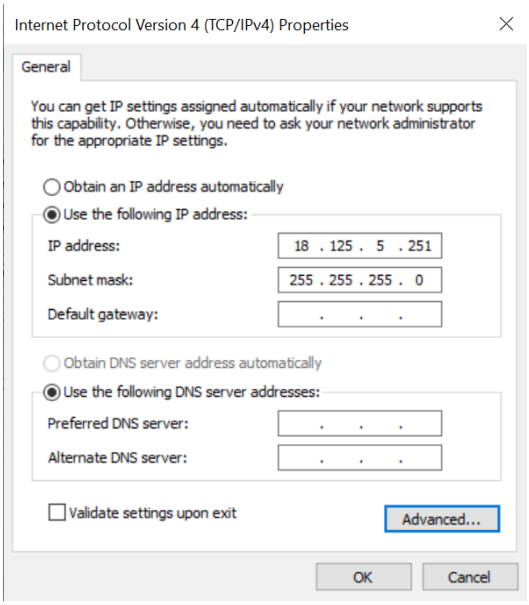


**Figure 4. IPv4 properties configuration**

7. Enter the IP address and its Subnet mask. For example, the IP (IPv4) address is 18.125.5.251 and the subnet mask is 255.255.255.0.
8. Click **OK**.

## 4.2 Board configuration

To connect the board by assigning an IP address, perform the following steps:

1. On the board, use the following command to create a new Ethernet wired network:
   `$ cat <<"EOF" >> /lib/systemd/network/80-wired.network`
   The newly created prompt (Ethernet network) is opened.
2. Navigate to the newly opened prompt for configuring the IP address.
3. Copy the IP address provided in the below example or use a different IP address:
   ***Note:*** *The IP addresses of the PC and the board are configured within the same IP network. Ensure that the Ethernet cable is connected between the board and PC.*

```
[Match]
Name=eth0
KernelCommandLine=!ip

[Network]
Address=18.125.5.252/16
EOF
```

4. Reboot the board.
   ***Note:*** *If the Ethernet cable is connected, the IP address is automatically assigned.*
5. To verify whether the board is configured with the IP address or to retrieve the board IP address, perform the following steps:
   a. Run the command `ifcommand`.
   b. Search for `eth0` configuration.
      For example, `eth0` is configured with IP address 18.125.5.252 in this context.

# 5 Configuring Eclipse environments

This section describes installation, setting up, and debugging using the Eclipse IDE.

In addition, this section describes the usage of Eclipse IDE developed using multiple programming languages, such as C and Python. The programming languages include specific installations and usage examples.

## 5.1 Configuring IDE using C language

This section describes the steps to download and install Toolchain, and Eclipse IDE packages. In addition, it describes the process to set up the Eclipse IDE using C as the programming language.

### 5.1.1 Installing Toolchain

C is a compiled language and its applications for the i.MX 8M board architecture involves cross-compilation. Toolchain is used to build the projects in C language.

Toolchain contains a set of tools, such as a compiler, linker, and assembler. These tools are required to build the projects in C/C++.

To install Toolchain, perform the following steps:

1. Download the Toolchain installer package for Windows by using the link given below:
   `gcc-arm-10.3-2021.07-mingw-w64-i686-aarch64-none-linux-gnu.tar.xz`
   ***Note:*** *The Windows version 10.3-2021.07 is used in this context.*
2. After downloading the package, extract the files of the installer package.
   For example, extract the files using the application 7Zip.
3. Launch the setup and install the Toolchain.
   ***Note:*** *Prefer to download in the default location.*

### 5.1.2 Installing Eclipse IDE

This section explains the steps to download and install the Eclipse IDE and configure the workspace.

To install the Eclipse IDE, perform the following steps:

1. Download the Eclipse package using the link given below:
   `Eclipse IDE for Embedded C/C++ Developers`
   For example, the Eclipse version 12.2021 is used in this context.
2. Open the downloaded folder.
3. Create a workspace or use a desired workspace.
   For example, the workspace `imx8-C-eclipse-workspace` is used this context.

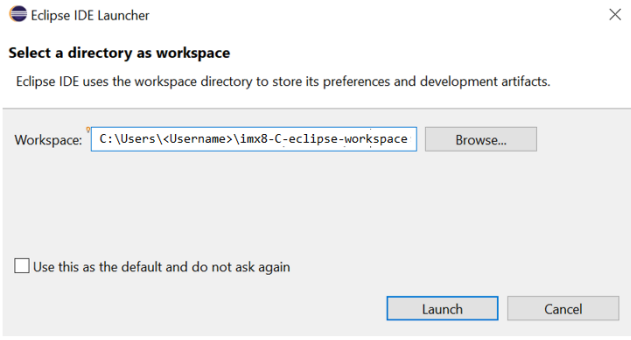


**Figure 5. Eclipse workspace**

#### 5.1.2.1 Workspace selection

To create a C/C++ project, perform the following steps:

1. Open the Eclipse IDE UI.
2. Navigate **File > New**.
3. Select **C/C++ Project**.
   A new project window **C/C++ Project** opens.

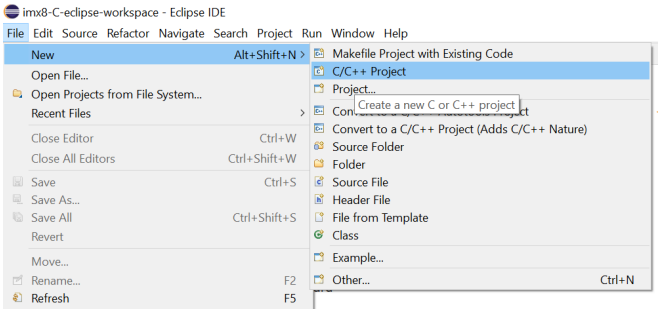


**Figure 6. New C/C++ Project**

### 5.1.3 Sample program

To create a sample `Hello, World!` program in C or C++, perform the following steps:

1. Navigate to **C/C++ Project > New Templates for New C/C++ Project**.
2. In the left pane, click **All**.
3. Select **C Managed Build** or **C++ Managed Build** depending on your build requirement.

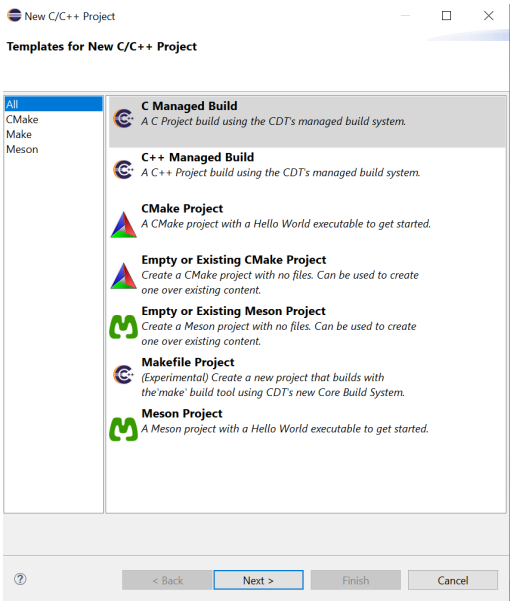


**Figure 7. C managed build**

4. Click **Next** and navigate to the corresponding project dialog box.
   In this context, **C Project** window appears.

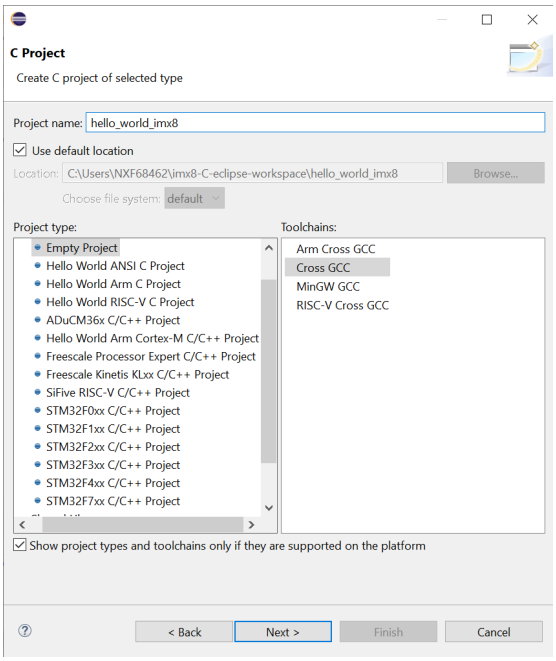


**Figure 8. Project type and toolchains**

5. Assign a name for the project.
6. In the **Project Type** section, select the **Empty Project** option.
7. In the **Toolchains** section, select the **Cross GCC** option.
8. Click **Next**.
   The **Select Configurations** window appears.
9. In the **Select Configurations** window, do not change the default settings.
10. Click **Next**.

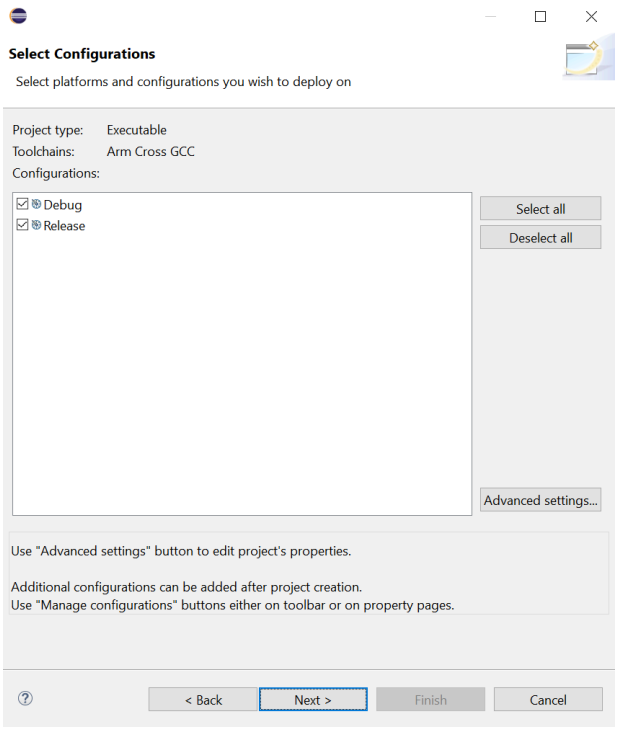


**Figure 9. Select configurations**

11. In the **Cross GCC Command** tab, the Toolchain appears.
    ***Note:*** *The Toolchain* `aarch64-none-linux-gnu-gcc` *displays in this context.*
12. Add prefix in the **Cross compiler prefix** as shown in Figure 10.


**Figure 10. Cross GCC command**

    The **Cross compiler path** is pointed to the unzipped Toolchain directory `/bin`.
    Verify whether the `bin` directory contains the executable file `aarch64-none-linux-gnu-gcc`.
    The path used in the example Figure 10:
    `C:\Users\<User>\Documents\AN_Eclipse_VS\Toolchain\gcc-arm-10.3-2021.07-mingw-w64-i686-aarch64-none-linux-gnu\bin`
13. Add **Cross compiler prefix** and **Cross compiler** path details.
14. Click **Finish**.
    If the project window does not appear, then perform the steps from step 15 to step 19.
15. Click the highlighted icon in the left pane of the screen:

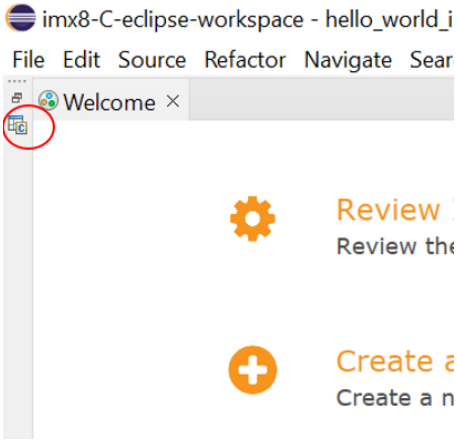


**Figure 11. Project Window icon**

16. Open the **Project Explorer** tab and create a source folder **src** for your project.
17. Click **Finish**.

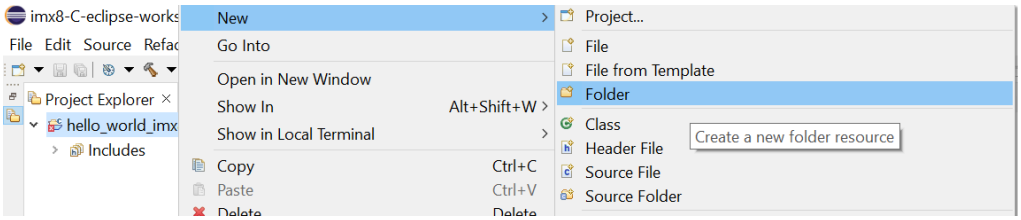


**Figure 12. New folder**

18. Add the new file to the `src` folder.

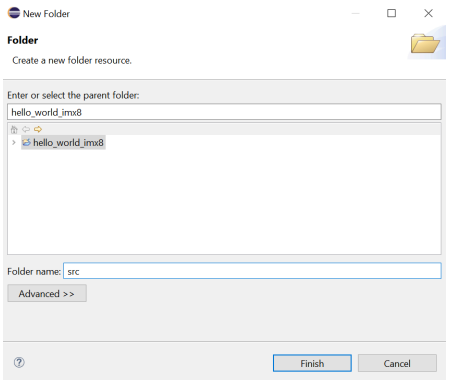


**Figure 13. `src` folder**

This new file is the source of `Hello, World!`. Ensure that the extension `*.c` is added for the C projects and `*.cpp` for the C++ projects.

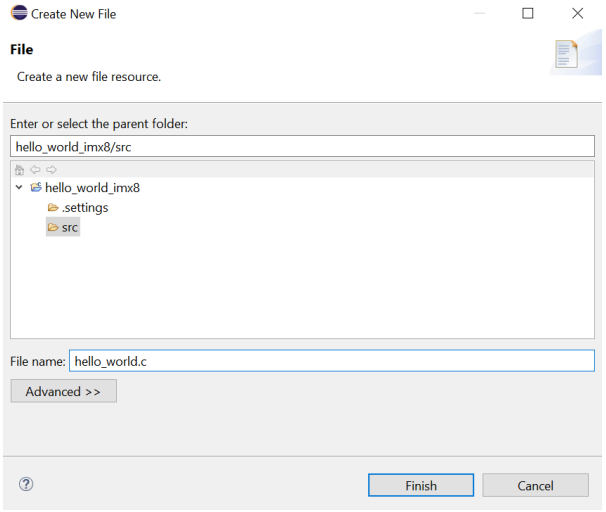


**Figure 14. New file**

19. Write a simple program for **Hello, World!** using the steps given below:
    a. In the Notepad application, open a new blank file.
    b. Write the steps as shown in Figure 15 and Figure 16.
    c. Save it with a new filename `hello_world` by adding an extension `*.c`.

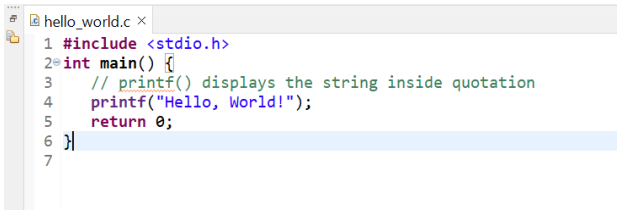


**Figure 15. "Hello, World!" example program in C**

For C++ project, save the file by adding an extension `*.cpp`.

```
#include <iostream>

using namespace std;

int main()  {
    cout << "Hello, Worldplusplus!" << endl;
}
```

**Figure 16. "Hello, World!" example program in C++**

    d. To build the debug version of the executable file, click the **hammer** icon.
    ***Note:*** *Do not receive any errors on cross-compilation.*

#### 5.1.3.1 Debugging image configurations

This section provides the process to debug the configurations created in C/C++.

To debug the Linux image configurations, perform the following steps:

1. From the menu bar, navigate to **Run > Debug Configurations** to debug the configurations.
   ***Note:*** *A similar* ***Debug Configurations*** *window appears on the screen.*

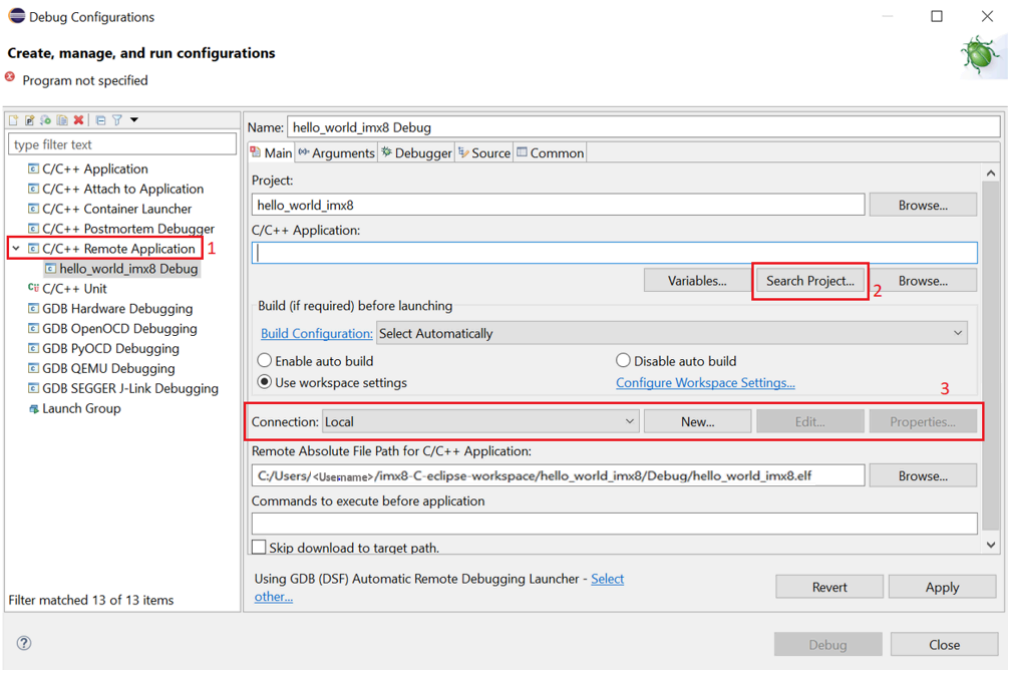


**Figure 17. Debug configurations window**

2. In the **Debug Configurations** window, select **C/C++ Remote Application** as shown in Figure 17.
3. Under **C/C++ Application**, click **Search Project**.
4. Search for the project binary file `hello_world_imx8`, select it, and then click **OK**.
5. To connect to the board, perform the following steps:
   a. From the **Connection** section, click **New**.
   b. Select **SSH**.
   A sample tab **New Connection** appears as shown in Figure 18.

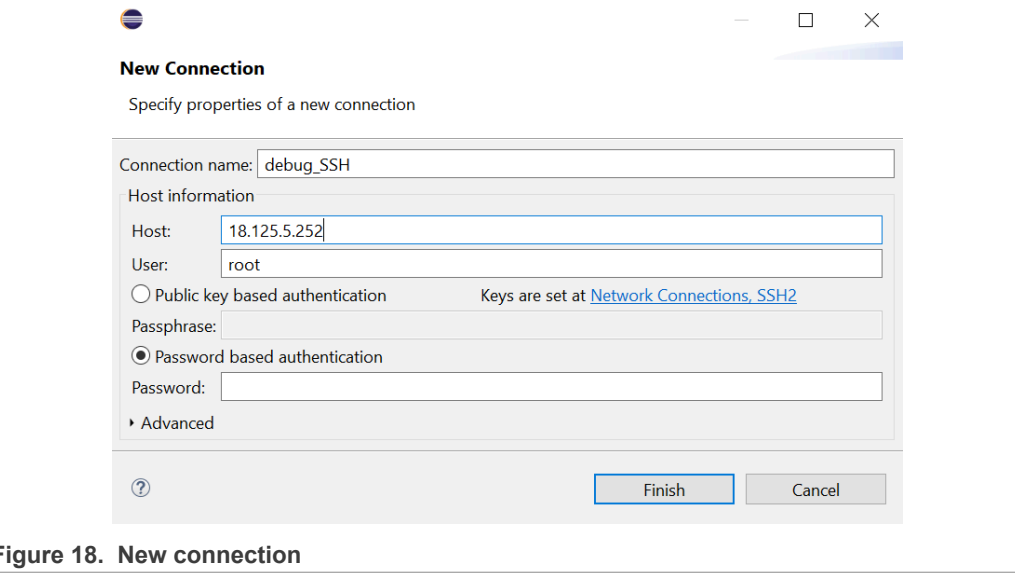


**Figure 18. New connection**

6. In the **New Connection** tab, provide the details in the fields as given below:
   - **Connection**: Enter the desired connection name.
   - **Host**: Enter the host address. For example, 18.125.5.252

- **User**: Enter the user name. For example, `root`.
  ***Note:*** *Select* ***Password based authentication****.*
- **Password**: Enter password for the user name `root`.

7. Click **Finish**.

#### 5.1.3.2 Debugging SSH configurations

To debug the SSH configurations, perform the following steps:

1. Navigate to the **Debug Configuration** window.
2. In the menu bar, navigate **Run > Debug Configurations**.
3. In the **Remote Absolute File Path for C/C++ Application** text box, choose a name for the debug executable file to upload on the board. For example, `debug_eclipse`. Figure 19 shows the Remote Absolute File Path section of the **Debug Configurations** window.

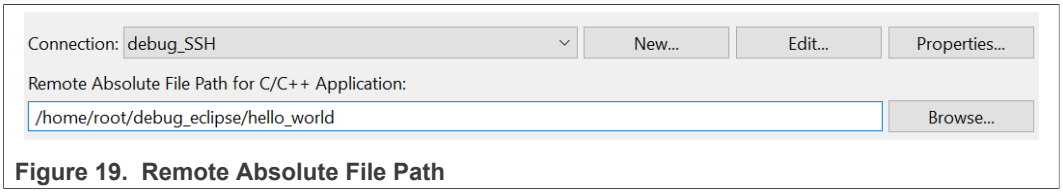


**Figure 19. Remote Absolute File Path**

4. On the **Debugger Options** section, navigate to the **Main** tab.
5. From the **GDB debugger**, click **Browse** to select the debugger file.
   The debugger file is the associated file of the GCC toolchain. For more information, see Section 5.1.1.

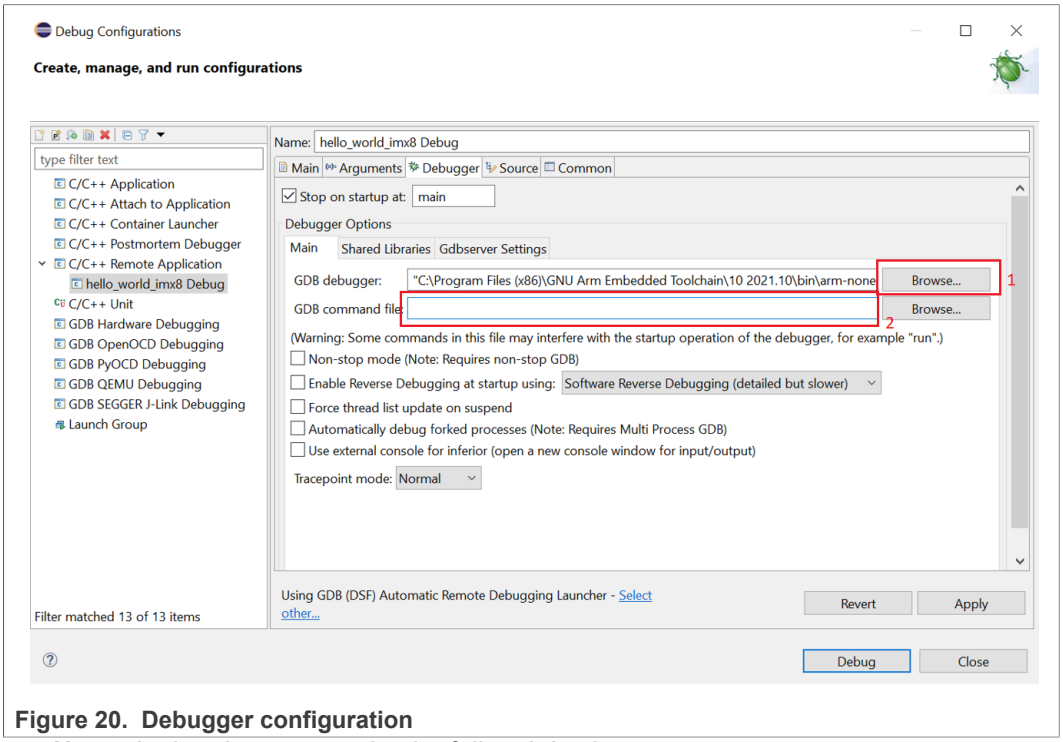


**Figure 20. Debugger configuration**

***Note:*** *In the above example, the full path is given as* `C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10\bin\arm-none-eabi-gdb.exe`*. A similar path is available if the toolchain is installed in its default location.*

6. Remove the contents from the field **GDB command file**.
7. Click **Debug**.

***Note:*** *If you are connecting for the first time to the board, then a pop-up message regarding the authenticity of the IP host appears.*

8. To continue connecting, click **Yes**.
   A file `debug_eclipse` appears on the board console.
   An error shown in Figure 21 displays on the Eclipse UI.
   **GDB version error information**:
   - Occurs due to the GDB cross-compiler and does not affect the running and debugging of the program.
   - Does not have many functionalities and it is similar to the error which occurs during compilation on Windows.
   - One of the lacking functionalities of this error is its version which is reported in the Windows compiler command.

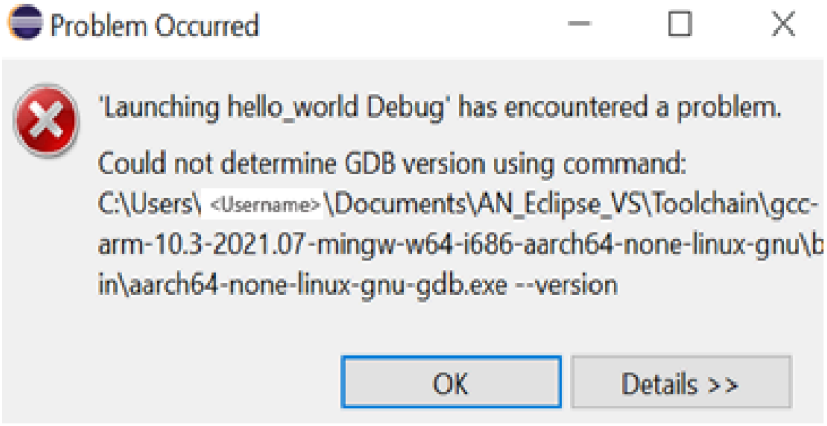


**Figure 21. GDB version error**

9. Ignore the error and click **OK** to continue. The example file is accessible to run from the Eclipse console. It depicts the board console through SSH.
10. Click the **Run** button to run the debug executable file.

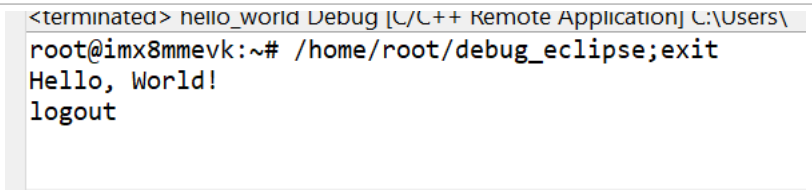


**Figure 22. Running output**

11. To check whether the file is successfully displayed on the board console, use any of the following ways:
    - Run the debug version of the executable file on the board.
    - Check whether the file is cross-compiled for the i.MX 8M platform using the following commands:

```
$ ls
$ file debug_eclipse
$ ./debug_eclipse
```

```
root@imx8mmevk:~# ls
debug_eclipse
root@imx8mmevk:~# file debug_eclipse
debug_eclipse: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-aarch64.so.
1, for GNU/Linux 3.7.0, with debug_info, not stripped
root@imx8mmevk:~# ./debug_eclipse
Hello, World!
root@imx8mmevk:~#
```

**Figure 23. Cross-compilation check on the board console**

12. If any errors are encountered upon running from Eclipse, set the full **Gdbserver** path in the **Gdbserver Settings** tab as shown in Figure 24:

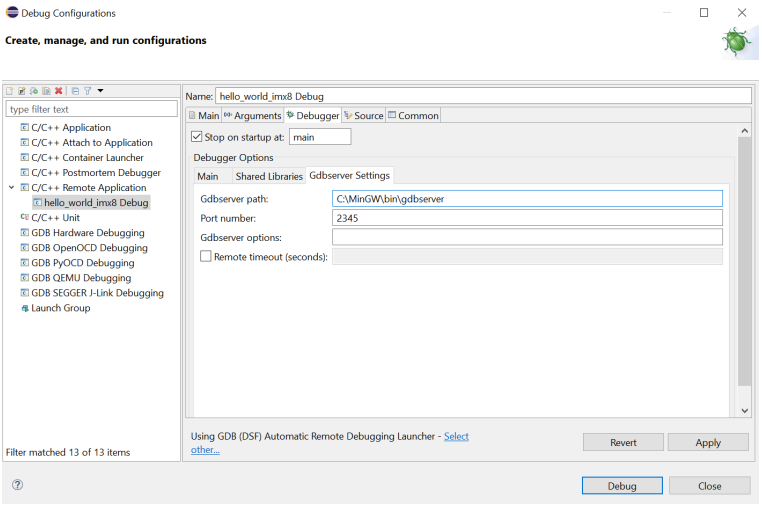


**Figure 24. Full gdbserver path**

## 5.2 Configuring IDE using Python language

This section describes the steps to install Eclipse IDE and set up Python. In addition, it describes the steps to transfer a file directly from the Eclipse IDE to a remote system.

### 5.2.1 Installing Eclipse for Java and Python setup

To install Eclipse and configure Python, perform the following steps:

1. Download Eclipse for Java and open it.
   For example, the Eclipse version 10.2021 is used in this context.
2. Create a workspace or use a specific workspace.
   For example, the workspace `imx8-python-eclipse-workspace` is used in this context.

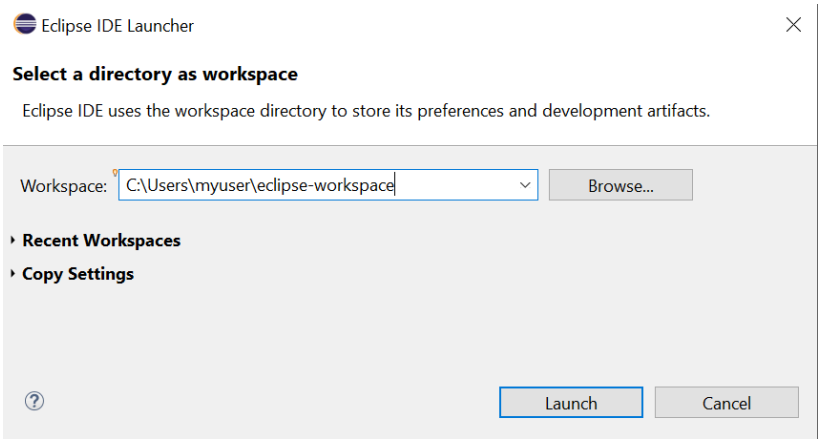


**Figure 25. Workspace selection**

3. From the Menus bar of Eclipse, navigate through **Help > Eclipse Marketplace**. The plug-ins for Python development and Remote Systems Explorer are added from this location.
4. Navigate to the **Search** tab.
5. In the **Find** search bar, enter **pydev**, and click **Go**.
6. Install the **pydev** package with all of its components. Figure 26 shows the installed Python IDE for Eclipse.

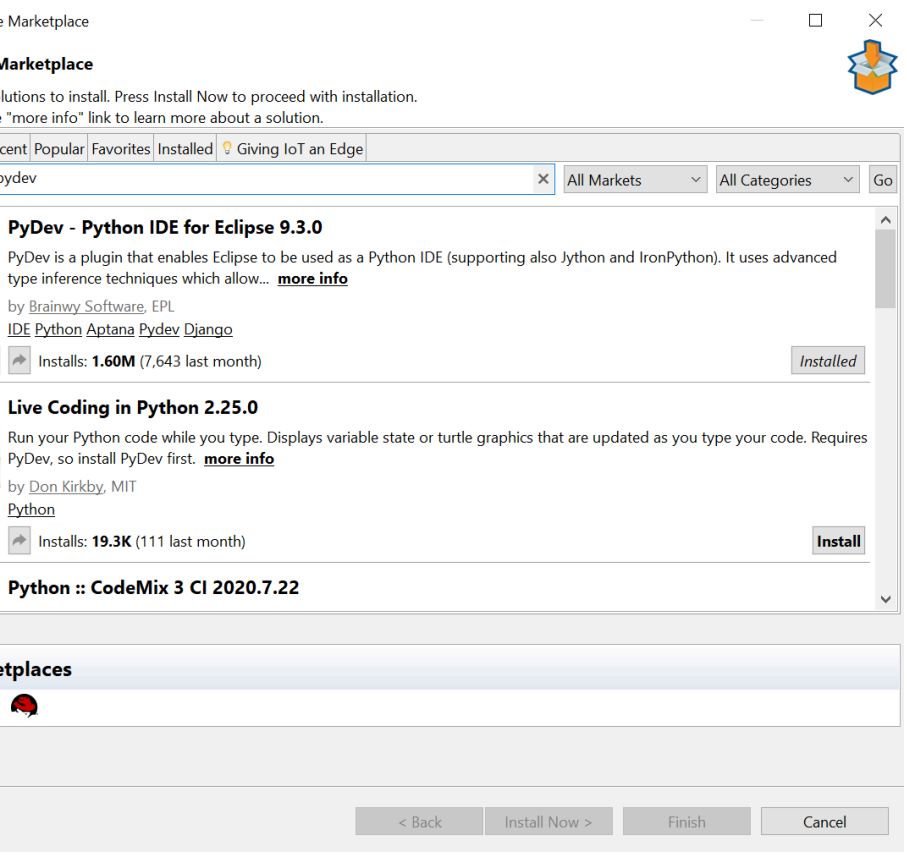


**Figure 26. Eclipse marketplace for PyDev**

7. After installation, restart Eclipse if prompted and navigate to the Eclipse UI.
8. For Remote Systems Explorer, navigate through **Help > Install New Software**.

9. From the **Work with** field, select **All Available Sites**.
10. Type **remote system** as the filter text to narrow down the list of sites.
11. Under the **General Purpose Tools** options, select **Remote System Explorer User Actions** to install.
12. Click **Next**.

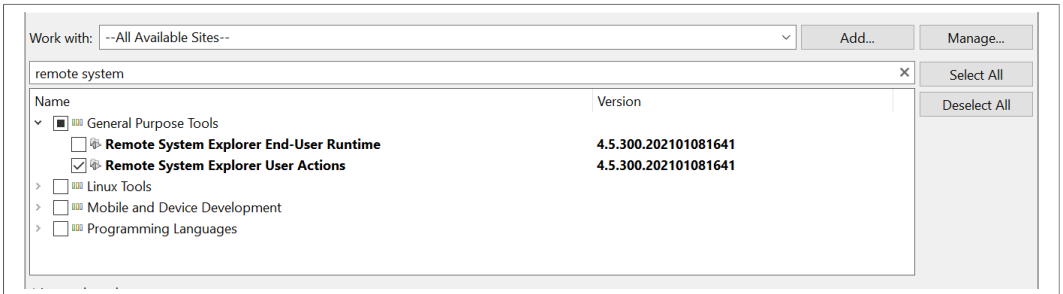


**Figure 27. Remote System Explorer**

13. After completion of the installation, restart Eclipse if prompted.
    To proceed further, ensure that the following requirements are met for the board:
    - The board is connected to the PC through an Ethernet cable.
    - The board has an IP address as described in the PC - board connection section of this document.
14. Click the Eclipse menu bar.
15. Navigate to **Window > Open Perspective > Other... > Remote System Explorer** to open the RSE perspective.
16. Right-click the **Remote Systems** tab and select **New Connection** as shown in Figure 28.

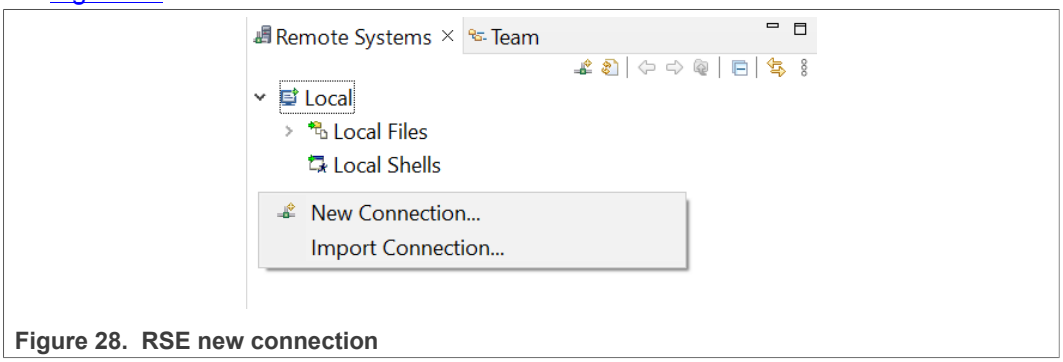


**Figure 28. RSE new connection**

17. Select **SSH Only** and click **Next**.
18. Complete the newly opened window by providing the board IP address.
    In Figure 29, the IP address 18.125.2.252 is provided in the **Host Name** field.
19. Provide a desired name for the newly created connection in the **Connection name** field as shown in Figure 29.

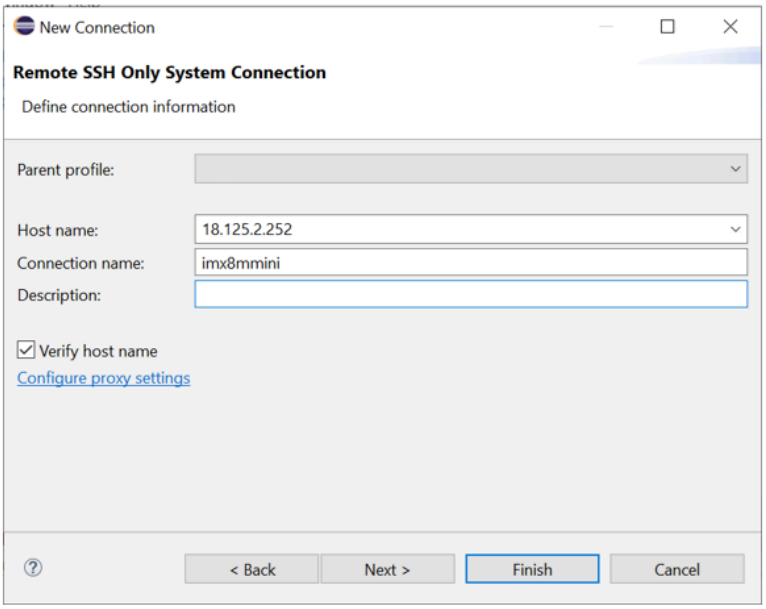


**Figure 29. New connection example**

20. Click **Finish** and proceed with the next steps.
21. From the **Remote Systems** tab, navigate to **Sftp Files > Root**.
22. In the pop-up window, provide the login credentials for the board.
    The default credentials are:
    - **User ID**: root
    - **Password**: There is no password.

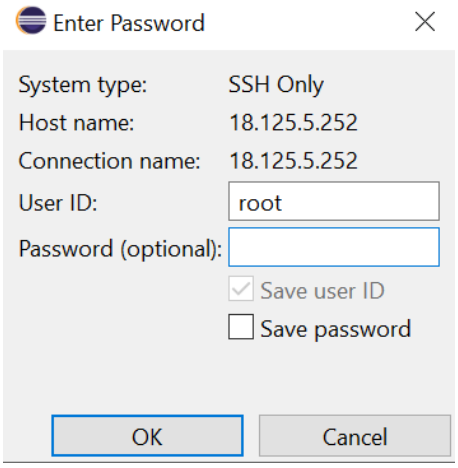


**Figure 30. Pop-up login window**

23. Click **OK**.
    You can view that the board is connected successfully from Eclipse using SSH. The file system is available from Eclipse through SFTP.
24. To create a folder on the board from Eclipse console, navigate to **home > root**.
25. Right-click the **root** folder.

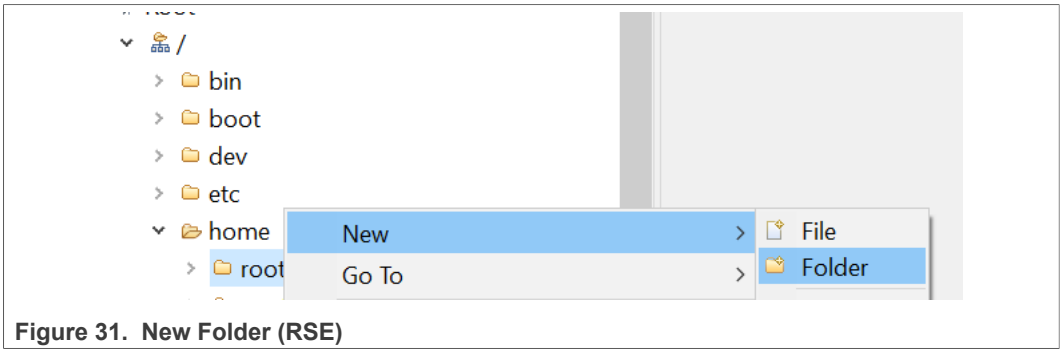


**Figure 31. New Folder (RSE)**

26. Enter a desired name to the folder and click **Finish**.
27. In the new working directory, create your first Python file `hello.py`.

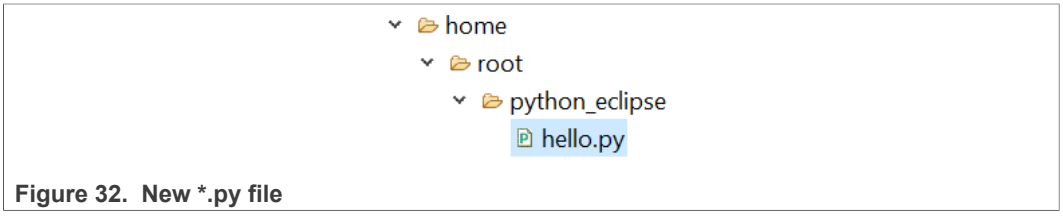


**Figure 32. New *.py file**

28. To configure the Python interpreter, select **Manual Config** as shown in Figure 33.

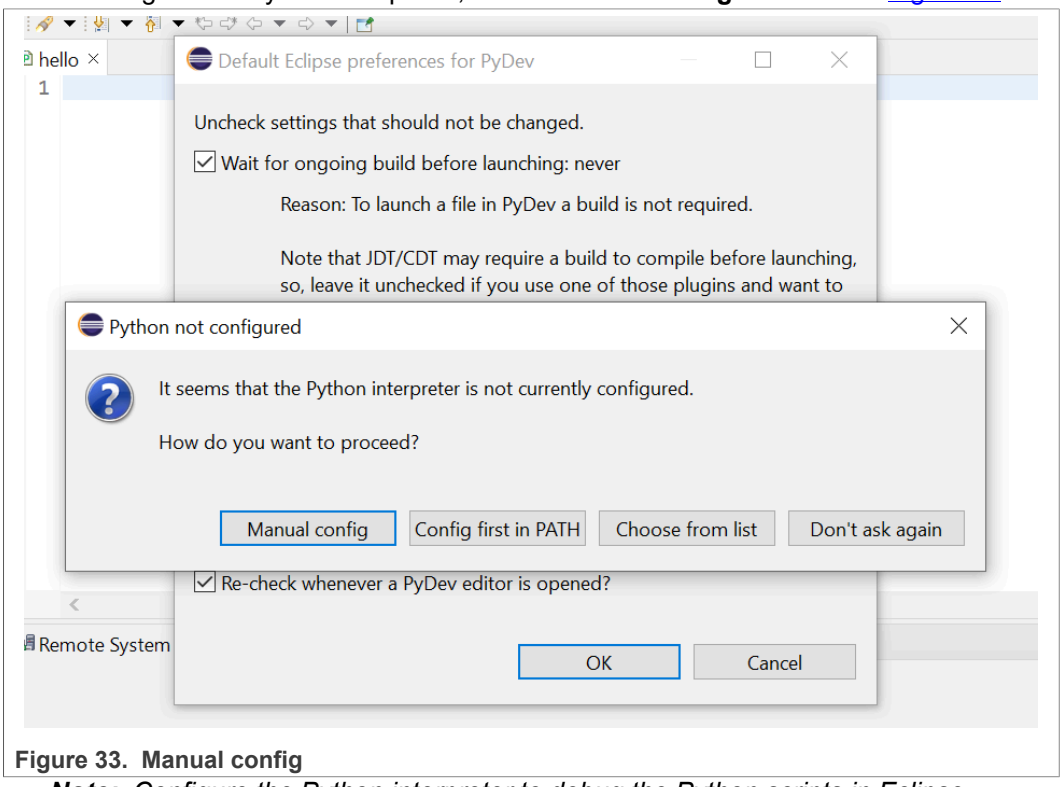


**Figure 33. Manual config**

***Note:*** *Configure the Python interpreter to debug the Python scripts in Eclipse.*

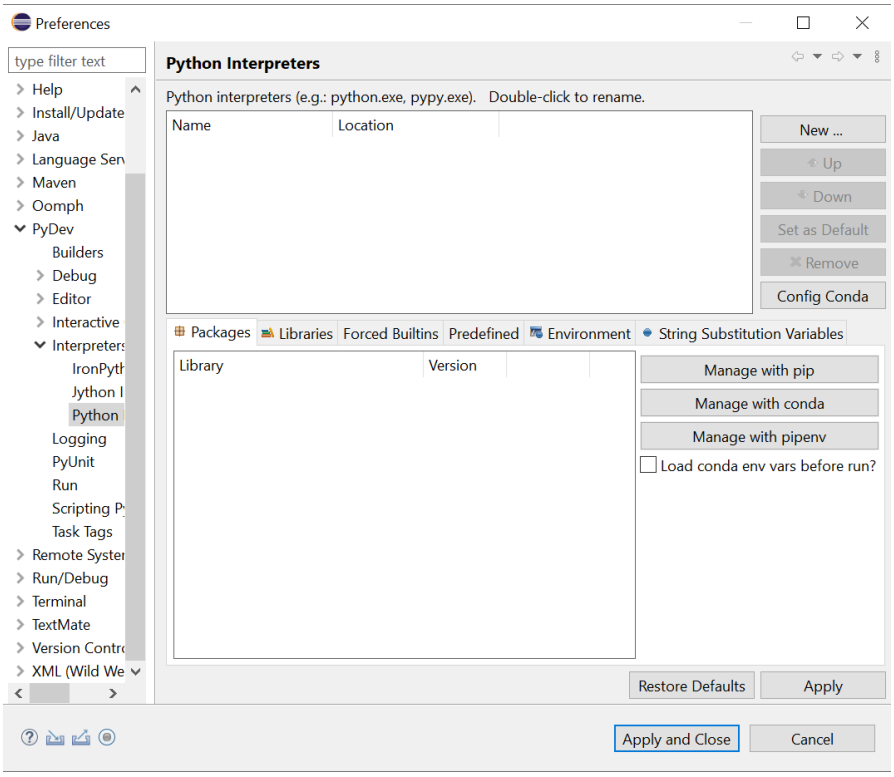


**Figure 34. Python Interpreter main window**

29. Install the Python interpreter on your host, if it is not installed.
    To check whether the Python interpreter is already installed on your host, perform the following steps:
    a. Open **Windows PowerShell**
    b. In the **cmd** window, type **python** as shown in Figure 35.

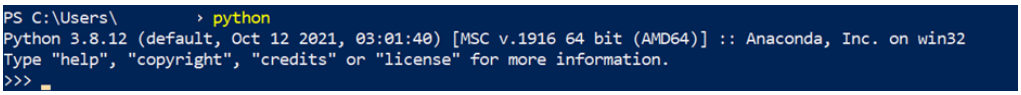


**Figure 35. Python version check**

***Note:*** *If Python is not found, then install it.*

30. If Python is installed on your host, continue the below steps.
31. Locate where Python is installed using the following command in the python prompt:

```
>>> import sys; print(sys.exec_prefix)
```

    The output should contain a path as shown in Figure 36.

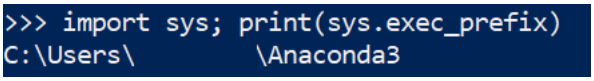


**Figure 36. Python path**

32. Navigate to the Eclipse console.
33. From the **Preferences** menu, navigate through **New > Choose** from Conda.
    ***Note:*** *For other Python interpreters, select the appropriate* ***Choose from...*** *setting.*
    In this example Figure 37, the interpreter used is **Anaconda3**.

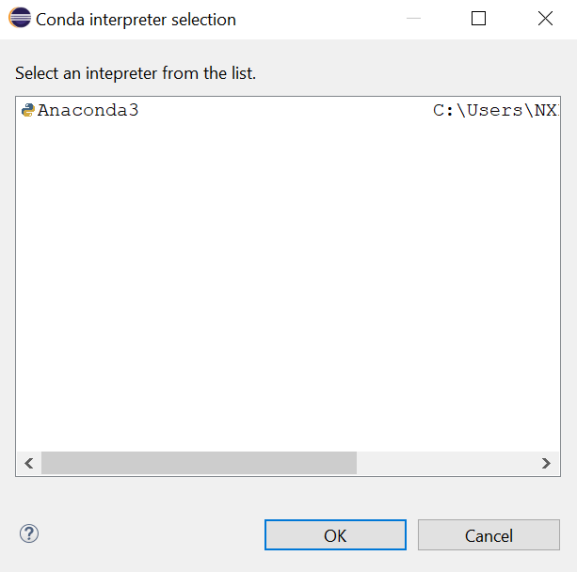


**Figure 37. Interpreter selection**

34. Click **OK**.
35. Do not change the default **PyDev** preferences as shown in Figure 38.

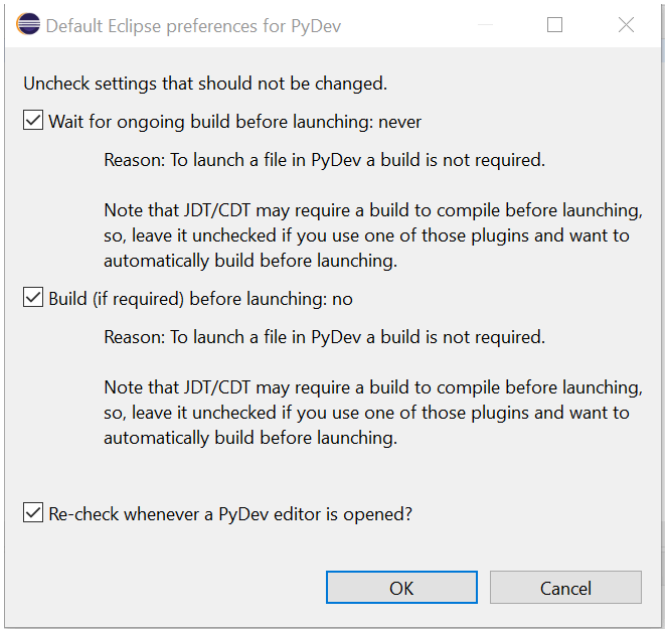


**Figure 38. Default PyDev preferences**

36. Click **OK**.

### 5.2.2 Sample program

This section describes how to write a sample `Hello, World!` program, which includes multiple prints, and demonstrate debugging. For more details, refer to Section 5.2.2.2.

To write a sample `Hello, World!` program, perform the following steps:

1. Navigate to the **Remote System Explorer** perspective.

If the perspective is not already set, then set the perspective by navigating from the menu bar through **Window > Perspective > Open Perspective > Other... > Remote System Explorer**.
In the RSE tab, the `python_eclipse` directory is created.

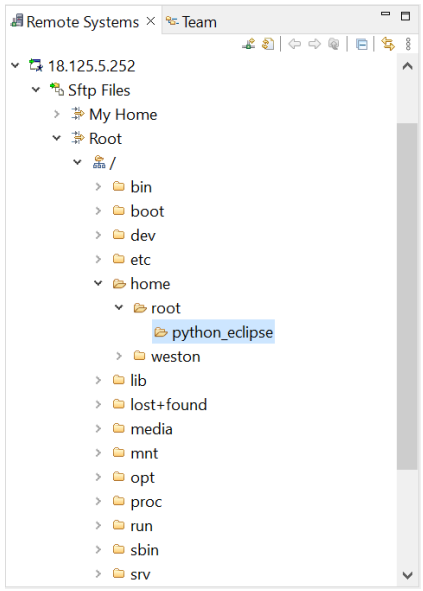


**Figure 39. Remote Systems tab**

2. Right-click the created directory and navigate to **New > File**.
   The below window appears:

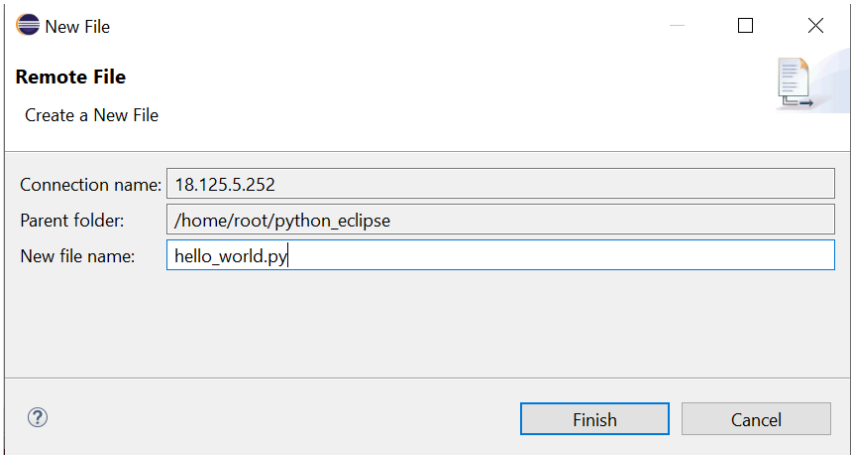


**Figure 40. Remote file creation**

3. Set a filename for the example file.
   Ensure that the `*.py` extension is included at the end of the filename.
   An example code is shown below:

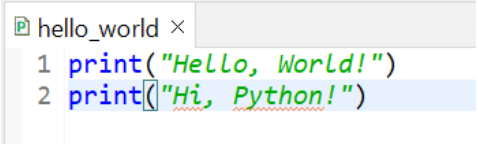


**Figure 41. Hello, World! example**

***Note:*** *Any example with multiple prints suffices.*

#### 5.2.2.1 Running the Eclipse configurations

This section explains the process to run the configurations.

To set a run configuration, perform the following steps:

1. Click the Eclipse menu bar and navigate to **Run > Run Configurations**.
   A sample running configuration is shown in Figure 42.

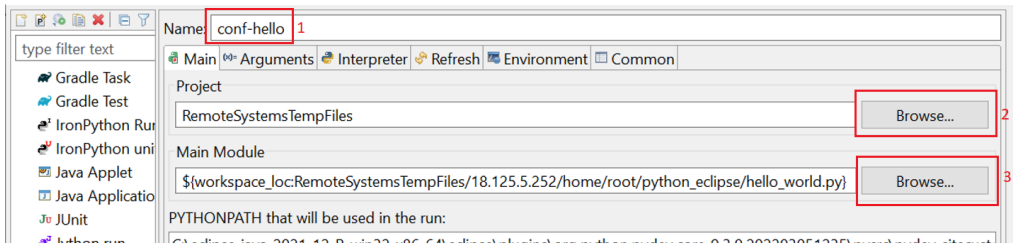


**Figure 42. Setting a run configuration**

2. Enter a desired name for the configuration. For example, `conf-hello` (outline 1).
3. In the **Project** section, browse for `RemoteSystemsTempFiles` (outline 2).
4. In the **Main Module** section, browse for the location of the `hello_world.py` file (outline 3).
5. Click **Apply** and **Run**.
   The file runs and prints the outputs in the **Console** tab as shown in Figure 43.

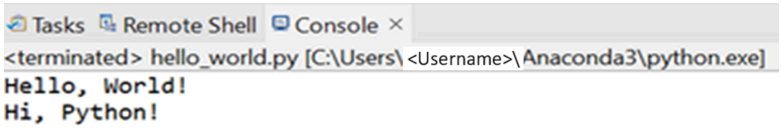


**Figure 43. Output example**

#### 5.2.2.2 Debugging

This section describes how to debug the configurations.

To debug the configurations, perform the following steps:

1. To set a breakpoint, double click the edge of the code window which is in front of the code line.
   A green mark appears as shown in Figure 44.

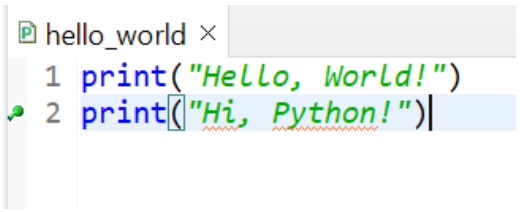


**Figure 44. Setting a breakpoint**

2. To start debugging, click the **Debug** icon.
   ***Note:*** *The same configuration that was used on running the program launches for debugging.*
   The following message appears which switches the perspective automatically to **Debug Perspective**.

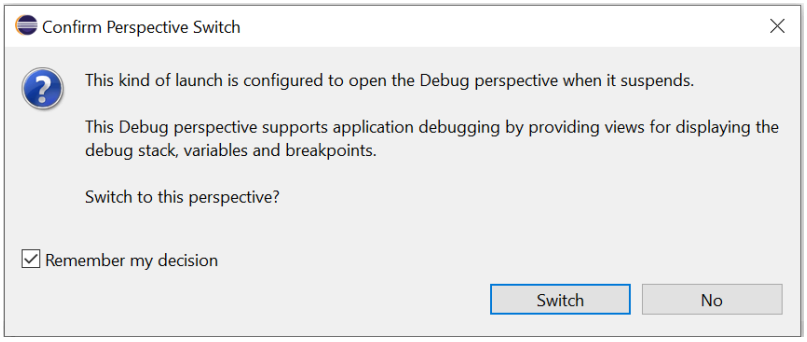


**Figure 45. Perspective switch**

3. Select **Switch**.
   If the message is not displayed, then change the perspective manually from **Window > Perspective > Open Perspective > > Other... > Debug**.

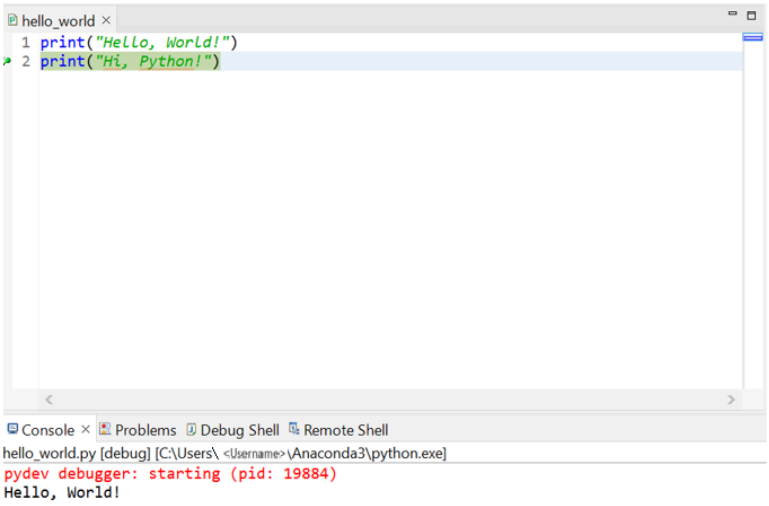


**Figure 46. Debugging steps**

4. To step over to the next breakpoint, press F6.

# 6 Configuring Visual Studio

This section describes the steps to download and install Visual Studio from Visual Studio 2022 Community. In addition, this section explains the steps to configure the remote SSH connection to the board.

## 6.1 Installing Visual Studio

1. Install Visual Studio using the link Visual Studio 2022 Community.
2. Navigate to the Visual Studio UI and select the hyperlink **Continue without code**.

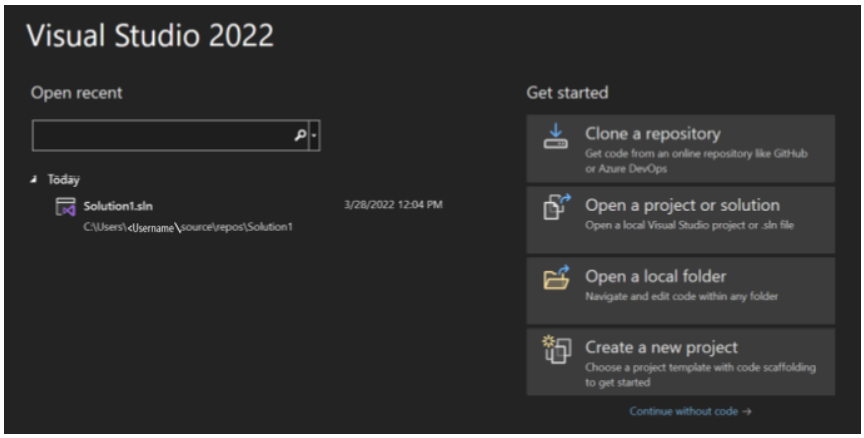


**Figure 47. Visual Studio**

3. Navigate from the menu bar to **Tools > Get Tools and Features...**
4. From **Desktop and Mobile**, select the **Desktop development with C++ toolkit**, and from **Other Toolsets**, select **Linux and embedded development with C++**.
5. Select **Modify** to install the selected packages.
   The IDE restarts to apply the changes.

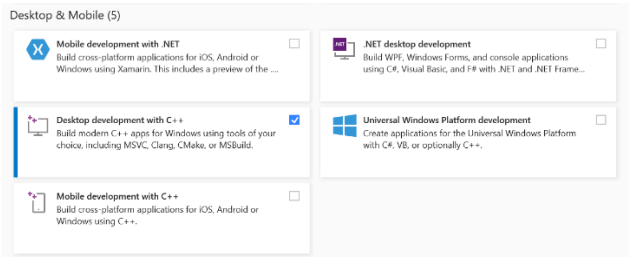


**Figure 48. Toolsets selection**

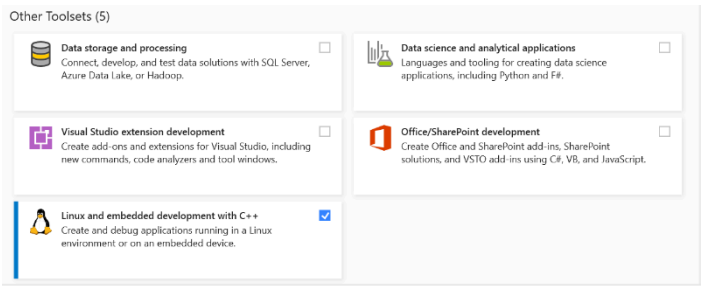


**Figure 49. Toolsets selection**

6. After restart, navigate from the menu bar to **Tools > Options**.
7. From the new dialog box, select **Cross Platform > Connection Manager**.
8. In the **Connection Manager** dialog box, choose the **Add** button as shown in Figure 50.

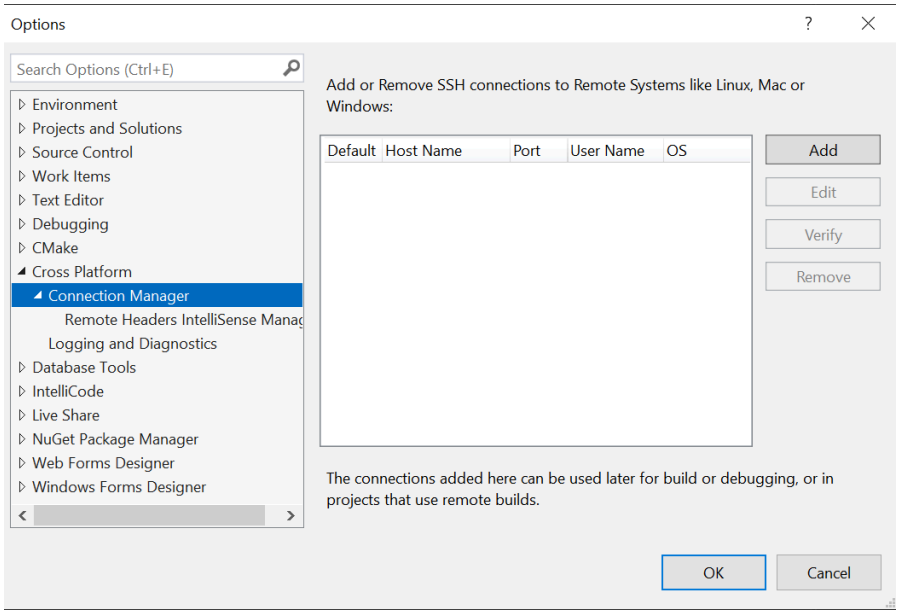


**Figure 50. Connection manager**

9. In the following window, enter the prompted details:
   - **Host name**: Enter the IP address of the board. For example, 18.125.5.252.
   - **Port**: Enter the port number. For example, 22.
   - **User name**: Enter the user name. For example, `root`.
   - **Authentication type**: Select the type of authentication. For example, `Password`.
   - **Password**: Enter the password for the user name `root`.

   An example is shown in Figure 51.

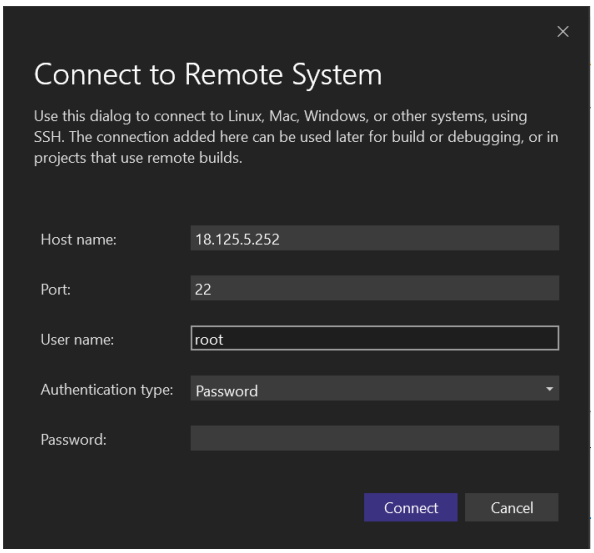


**Figure 51. Connect to remote system**

10. Select **Connect**. Now you have a remote SSH connection to the board.
11. Optionally, set the board connection as **Default** by selecting the radio button as shown in Figure 52.

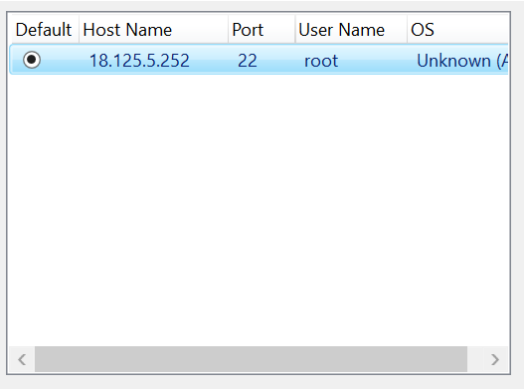


**Figure 52. Default board connection**

## 6.2 Sample program

This section explains how to create a sample `Hello, World!` program in C/C++.

To create a project in C, perform the following steps:

1. Navigate to **File > New > Project**.
   Ensure that you select **Linux** instead of **All Platforms.**
2. Select **Empty Project**.
3. Enter a desired name to the project. For example, `my_first_project`.

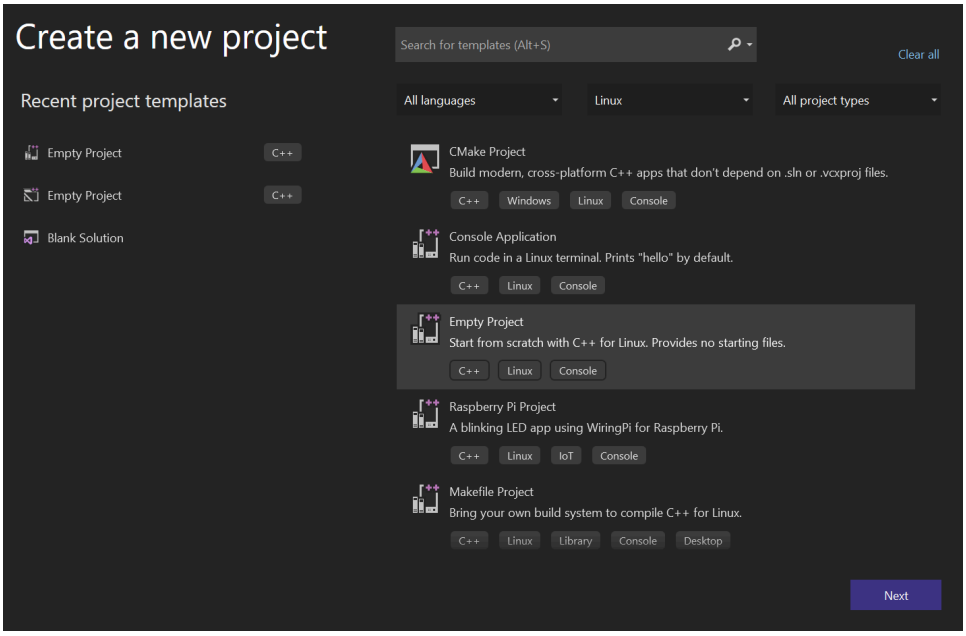


**Figure 53. New C/C++ Project**

4. To add files to the new project, navigate to the **Solution Explorer** window as shown in Figure 54.

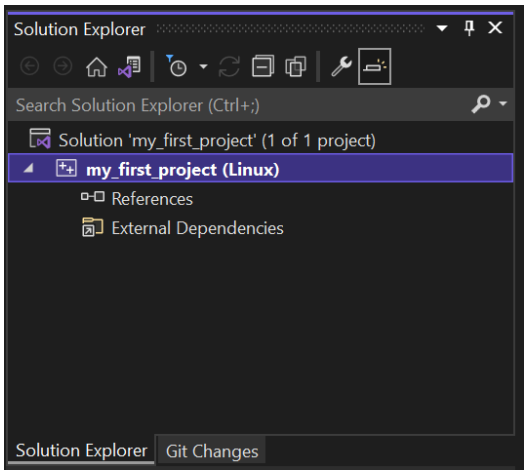


**Figure 54. Solution explorer window**

5. Right-click the new project. For example, `my_first_project` is used in this context.
6. Write a simple Hello, World! program in C++ as shown below:

```
#include <iostream>
using namespace std;
int main() {
 cout << "Hello, World!" << endl;
}
To run the program on the target, choose the Run button that
 has the IP address of the board next to it. It should look
 similar to the one below.
```

7. To debug on the board, include breakpoints through the program.
   For example, a breakpoint is included before the message output as shown in Figure 55.

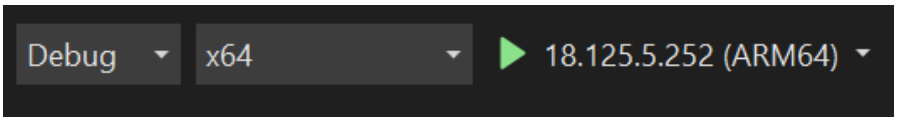


**Figure 55. Platform-specific Run**

8. Click the **Run** button again. The debug pop-up window appears as shown in Figure 56.

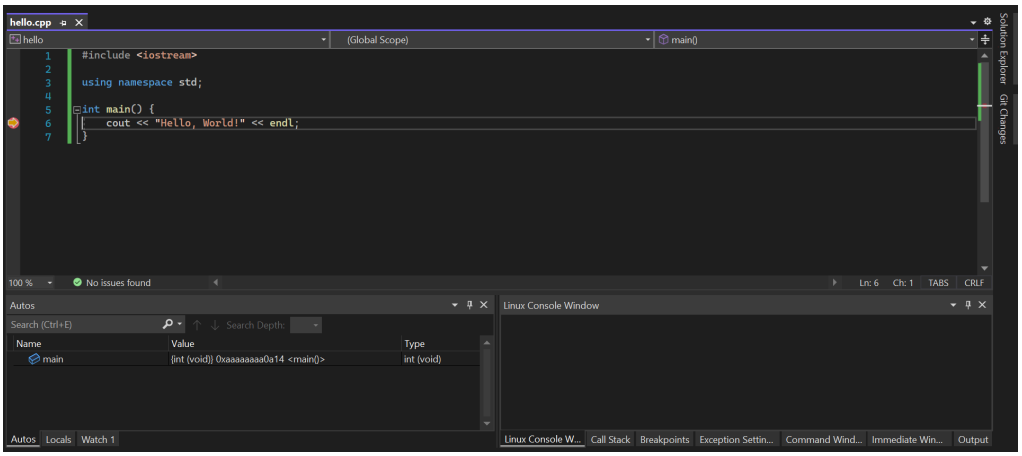


**Figure 56. VS debugging view**

9. For C development, use the steps from Step 1 to Step 8.
   Ensure that the source file extensions are changed to `*.c`.
   A sample C program `Hello, World!` is given below:

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

# 7 References

For more information on I.MX Devices, Python scripting, and Eclipse setup, refer to the following list of documents:

- i.MX Yocto Project User's Guide
- 
- i.MX Linux User's Guide
- Developing an Application for the i.MX Devices on the Linux Platform
- Raspberry pi development setup on Eclipse [Linux] and Debugging via SSH
- Remote development of Python scripts on Raspberry Pi with Eclipse

# Revision history

Table 3 summarizes the changes done in this document from the initial release.

**Table 3. Revision History**

| Revision number | Date | Substantive changes |
|---|---|---|
| 0 | 11 November 2022 | Initial release |

# 8 Legal information

## 8.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## 8.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

## 8.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

**AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile** — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

**i.MX** — is a trademark of NXP B.V.

# Contents





For more information, please visit: http://www.nxp.com

**Date of release: 11 November 2022**
**Document identifier: AN13739**