periCORE

Development Kit Setup

Application Note

Perinet
Seamless IoT Connectivity

Abstract

This application note guides developers and engineers in the initial setup of the *periCORE Development Kit*. It is formulated as a step-by-step guide, going through the hardware assembly, the software tools installation, as well as the basic configuration of the environment, in order to develop customer specific firmware applications for the *periCORE SPE communication module*.

Doc-No.: PRN.100.376 rev: 4 2025-02-26

Document Information

Title	periCORE
Subtitle	Development Kit Setup
Type	Application Note
Status	Release
Version	4
Date	2025-02-26
Disclosure Restriction	

Contents

1 Welcome

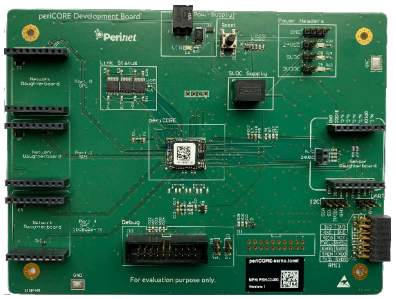
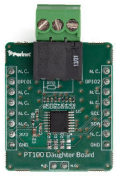
The *periCORE Development Kit* contains all the tools needed to program and test the *periCORE SPE communication module*. This application note will guide you through the steps needed to setup the hardware and the software environment. It provides an example on how to build and debug your own application and how to program the periCORE with the newly created application.

The objective of this guide is to provide a basic periCORE firmware development setup. For deeper reading, please see the documents suggested in Section 7.

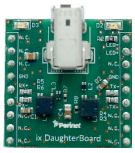
Prerequisites:

- periCORE Development Kit
- PC / Laptop (verified with Windows, macOS or Linux[1])
- 2 network ports (e.g. Ethernet switch) connected in your network
- 2 power sockets

The Development Kit contains the following components:

Item	Description	Product
1	1 pc. periCORE Development Board Article No. PRN.000.019	
2	1 pc. Daughterboard 0-10V Article No. PRN.000.026	
3	1 pc. Daughterboard Pt100 Article No. PRN.000.028	

[1] Verified with debian bullseye, Fedora 35|36|37 and Ubuntu 20-04

4	1 pc. Daughterboard GPIO Article No. PRN.000.029	
5	1 pc. Daughterboard T1IJ Article No. PRN.000.027	
6	1 pc. Daughterboard M8HM Article No. PRN.000.030	
7	1 pc. Daughterboard M8HF Article No. PRN.000.031	
8	1 pc. Daughterboard RJ45 Article No. PRN.000.032	
9	1 pc. Daughterboard IX Article No. PRN.000.033	
10	1 pc. Daughterboard MicroE SHT click-board Article No. PRN.100.536	

11	2 pc. Daughterboard T1H Article No. PRN.000.044	
12	1 pc. periMICA modular edge computer Article No. PRN.000.025	
13	1 pc. periLINE hybrid SPE cable Article No. PRN.000.017	
14	1 pc. periSTART standart Article No. PRN.000.003	
15	1 pc. periSTART power supply 24VDC Article No. PRN.000.035	

16	1 pc. J-Link Debugger Article No. PRN.100.132	
17	1 pc. M12 X-coded cable Article No. PRN.100.003	
18	1 pc. PoE injector incl. power supply & Ethernet cable (RJ45//RJ45) Article No. PRN.100.347	
19	1 pc. Distance sensor Article No. PRN.100.573	
20	1 pc. M12 sensor cable Article No. PRN.100.560	

Table 1: periCORE Development Kit Components

2 Architectural Overview

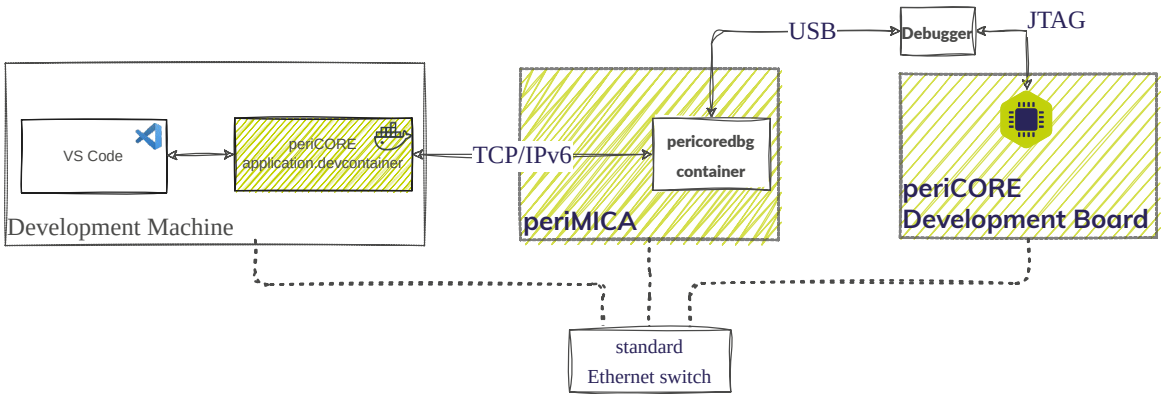


Figure 1: periCORE Development Kit Overview

The periCORE firmware development environment includes the deliverables of the periCORE Development Kit and an additional Development Machine (Windows, macOS or Linux), as shown in the system overview of Figure 1.

The Development Machine is not provided by Perinet. Perinet tested the periCORE Development Kit for Development Machines based on the platforms Windows, Linux as well as macOS. The setup procedure of the periCORE Development Kit is similar for all platforms.

2.1 Development Machine

A periCORE based firmware image is created during the cross-compilation process on the Development Machine. A dedicated OCI-conform container (periCORE application devcontainer [4]) is used to create the build-time artefacts. That container is provided by Perinet and publicly available. It includes the necessary tools (gnu based C++ compiler, linker, debugger, etc.) that are used at build-time to create a periCORE based firmware image. The final firmware download image includes the webUI.

Editing and cross-compilation of the source code is performed at the Development Machine. The created binary firmware image is uploaded via network into the RAM of the periCORE, in case of a debug-session, or can be uploaded via the RESTful API as part of periCORE's Life-Cycle process. The deployment of the firmware as well as the debugging is performed via network (TCP/IPv6).

2.2 Remote Debugging

A dedicated periCORE debug server periMICA-container [7] is the interface between the Devcontainer libperiCORE and the third-party debugger. It is executed on the periMICA and is pre-deployed delivered with the periCORE Development Kit. The periMICA container proxies the third-party USB-based JTAG debugger and allows to create a remote accessible development setup (debug server). Only a network connection to the periCORE debug server periMICA-container is necessary in order to debug a periCORE based firmware applications.

3 Quick Start

The periCORE Development Kit is designed for an easy and fast start to the development of periCORE based firmware. This section describes how to quickly set up the example periCORE based firmware application (periNODE-distance firmware [5]) for the periCORE Development Kit.

In summary the setup process is as follows: after assembling the periCORE Development Kit hardware, the periNODE-distance firmware [5] is fetched to the Development Machine. The firmware is then opened with the Devcontainer libperiCORE [4]. After cross-compiling the C++20 sources, the firmware is uploaded and debugged via a remote connection to the periCORE debug periMICA-container. The final application, can then be deployed in the field or at production via the RESTful API of the periCORE [6].

Note: For further details on the installation of the software prerequisites on the Development Machine see Section 5

3.1 Quick Hardware Assembly

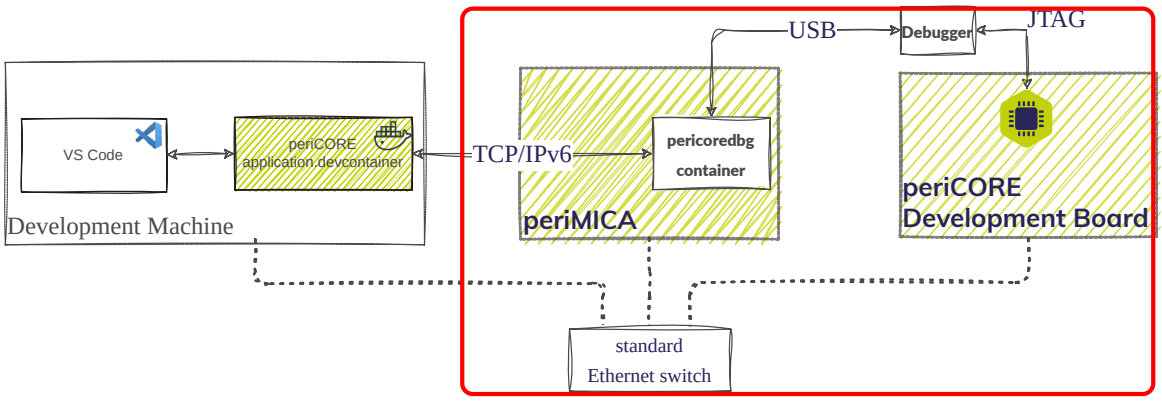


Figure 2: periCORE Development Kit Overview (Hardware Setup)

Setting up the hardware only of the periCORE Development Kit, abstract summarized with Figure 2, focused on the delivered components. The finalized setup of the periCORE Development Kit is shown in Figure 3. It connects standard Ethernet to the periMICA as well as to the periCORE Development Board. The latter is transformed via the periSTART to single pair Ethernet hybrid. It also provides 24V power supply to the periCORE Development Board on `Port0`.

Note: It is recommended to connect the standard Ethernet cables (RJ45 connector) of the periSTART and the PoE Injector to an Ethernet switch (not part of the periCORE Development Kit).

The third-party JTAG-debugger connects with USB to the periMICA and with a standard JTAG cable to the periCORE Development Board. The debugger is powered via periMICA's USB port.

The periCORE Development Board is delivered with previous deployed Daughterboards suitable to cover the use case of this document. The sensor side of the periCORE Development Board hosts the 0-10V Daughterboard, which connects via the M12 sensor cable to an ultrasonic distance sensor.

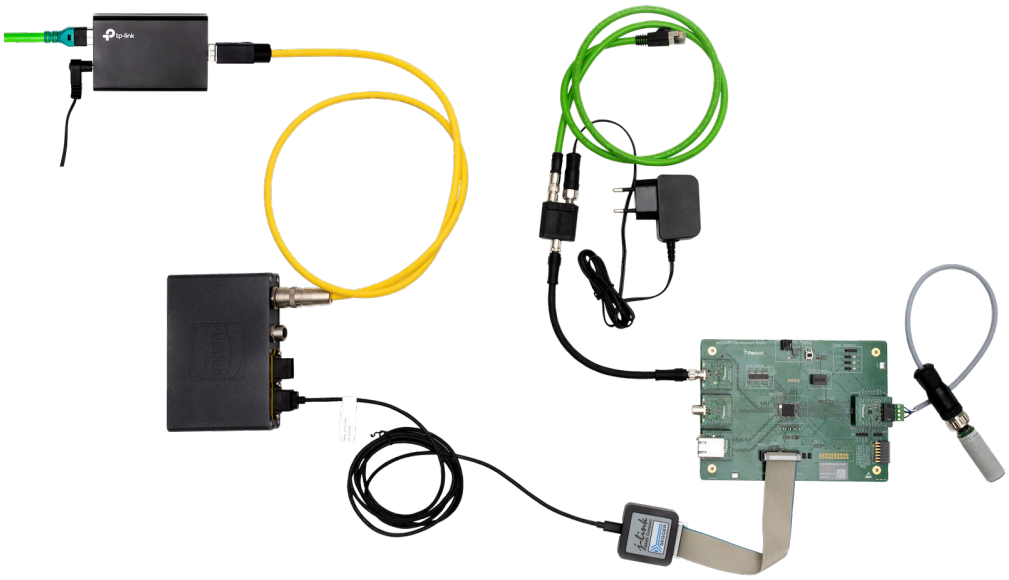


Figure 3: periCORE Development Kit complete setup

Note: More detailed information on the different step of the hardware setup is provided in Section 4

3.2 Quick Software Set Up

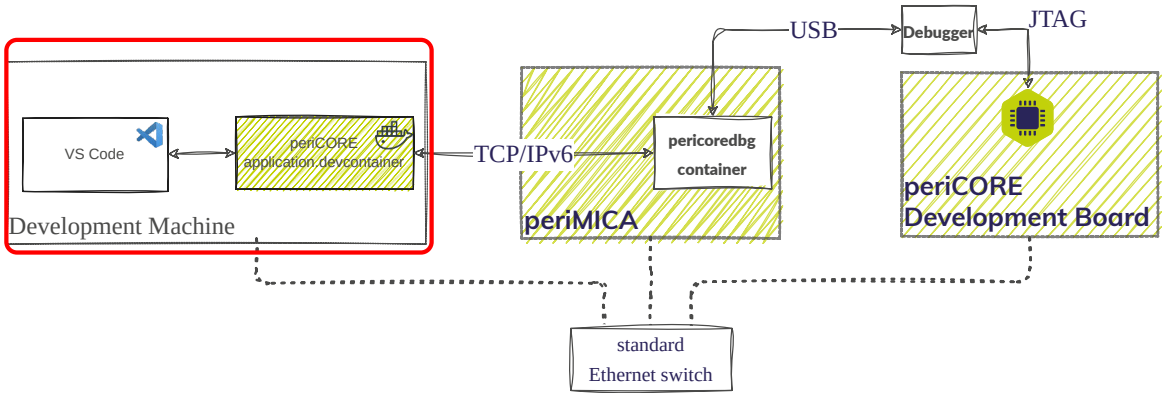


Figure 4: periCORE Development Kit Overview (Software Setup)

The Development Machine hosts the source code editor environment (Visual Studio Code) and the Devcontainer libperiCORE runtime (Docker). A prerequisite toolset is assumed to be installed on the Development Machine:

Docker Desktop a container runtime [14] that executes the prepared container *Devcontainer libperiCORE* [4], see Section 5.1

git-scm is a free and open source distributed version control system [3].

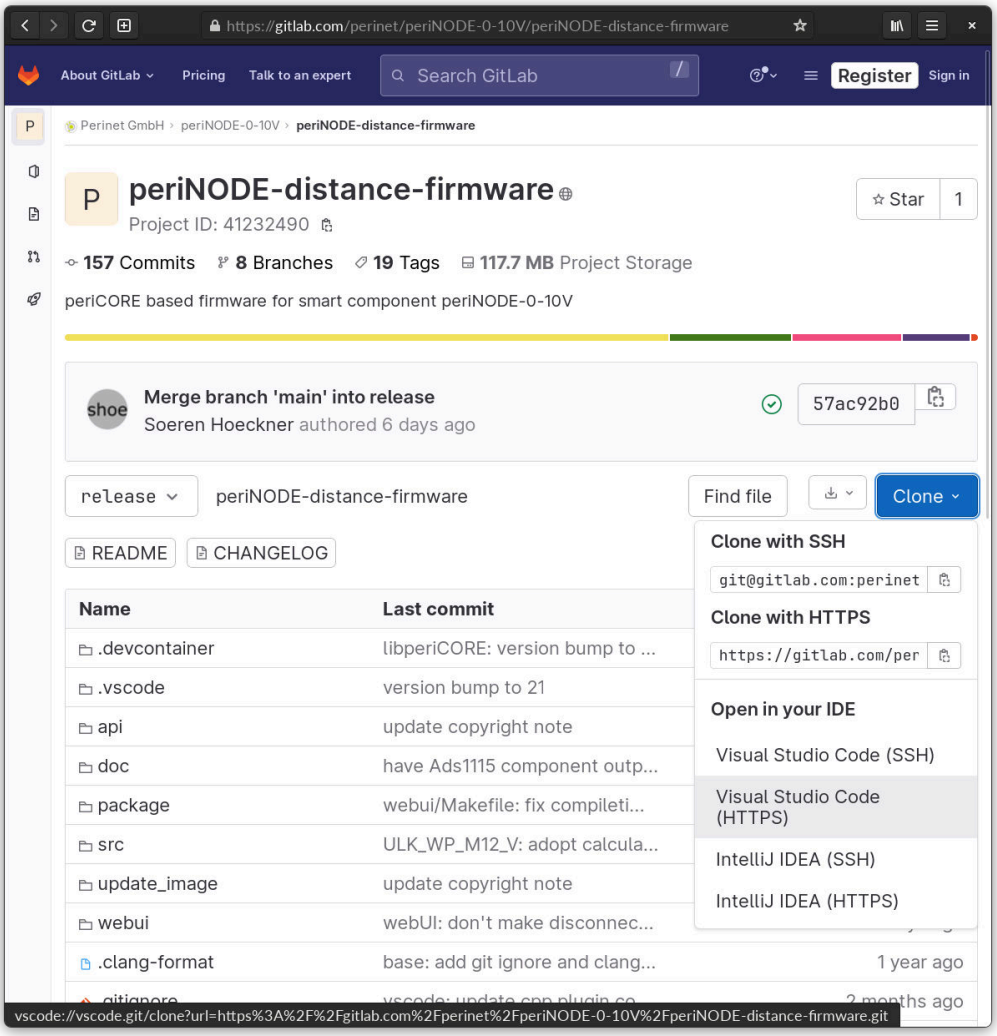
Visual Studio Code A plugin based source code editor [15], see Section 5.2.

Dev Containers Extension A Vs Code plugin to edit code with containerized toolchain [16], see Section 5.2.

An example firmware, the *periNODE-distance firmware*, has been prepared and is used as example for the periCORE Development Kit. The repository needs to be cloned to the *Development Machine* and opened with the libperiCORE Devcontainer [4]. This section gives a quick walk through to the necessary steps.

Note: At this point it is mandatory to have the prerequisites installed. Detailed installation instructions are documented in Section 5

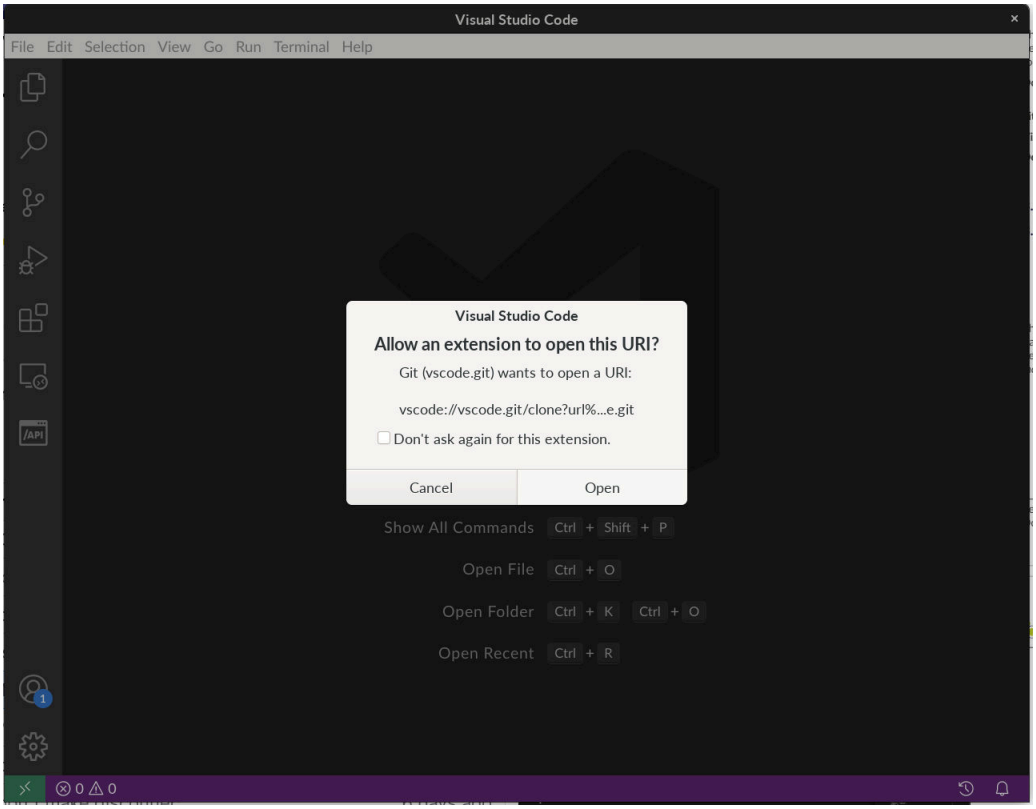
3.2.1 Browse to the periNODE-distance firmware Repository



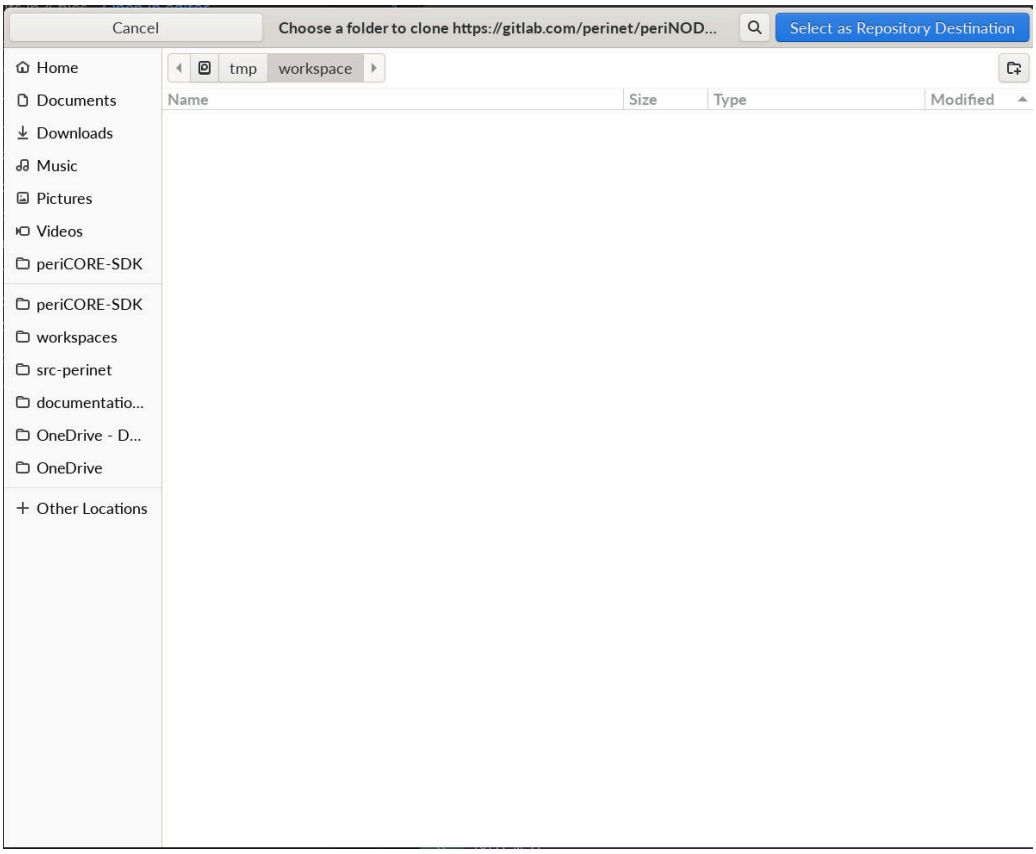
The *periNODE-distance firmware*[2] is available via GitLab.

[2] https://gitlab.com/perinet/periNODE-0-10V/periNODE-distance-firmware.git

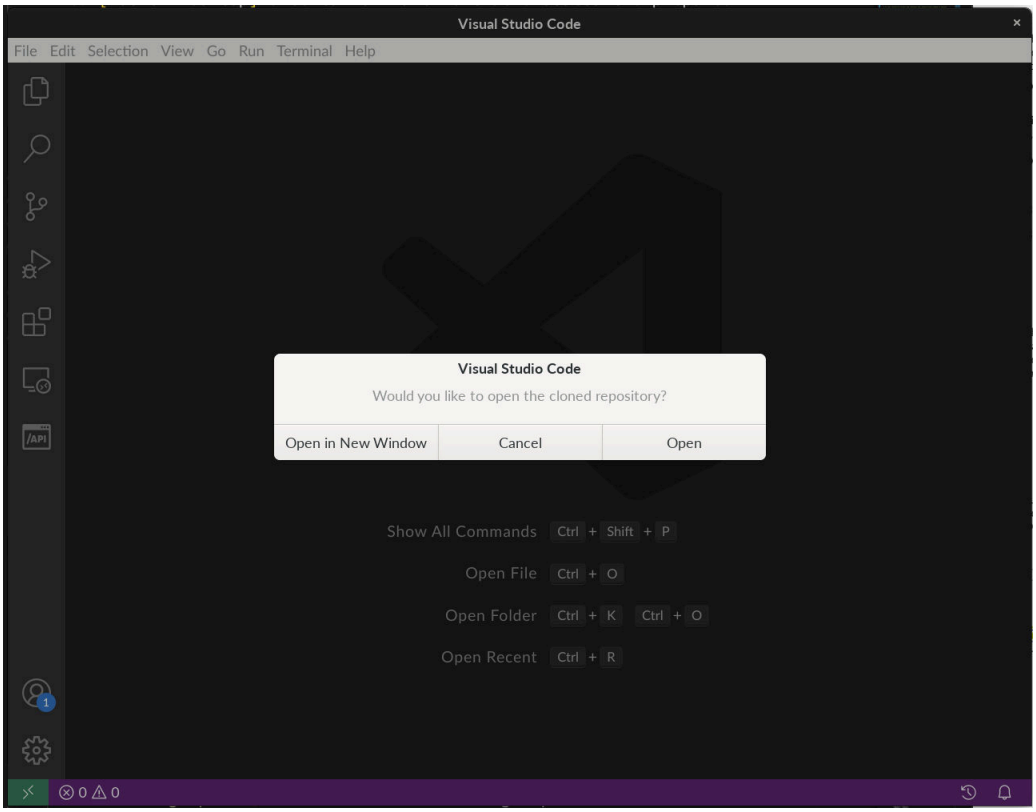
3.2.2 Open URL with Vscode



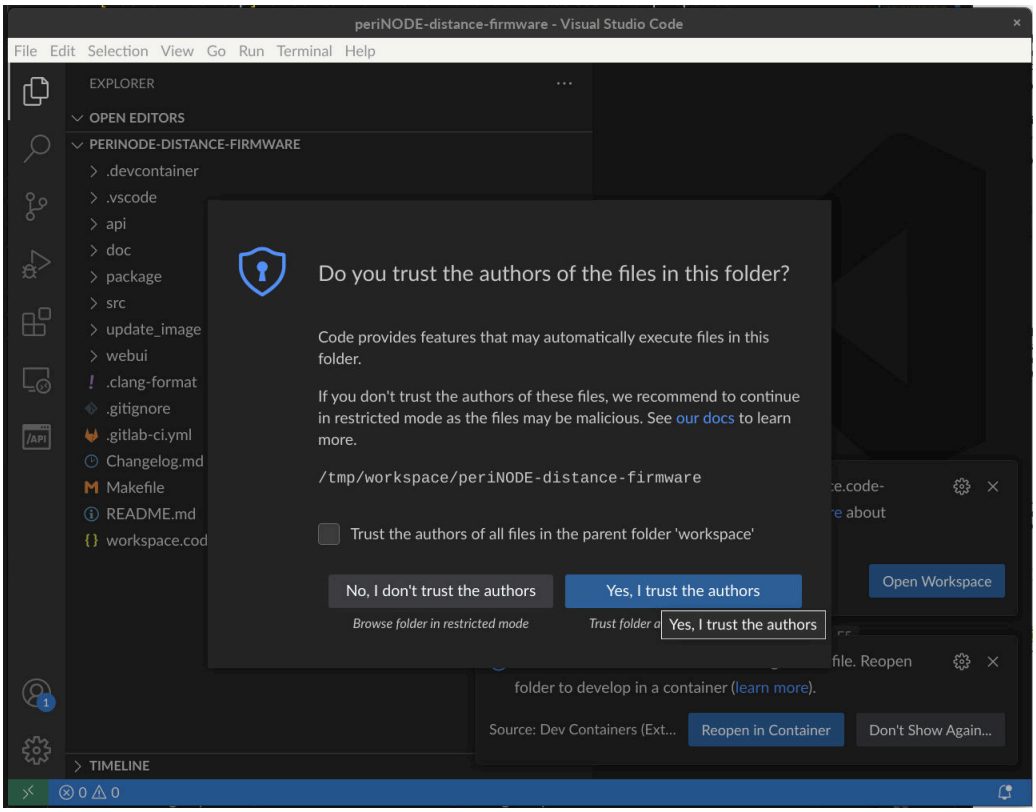
3.2.3 Select Parent Directory to Clone Repository



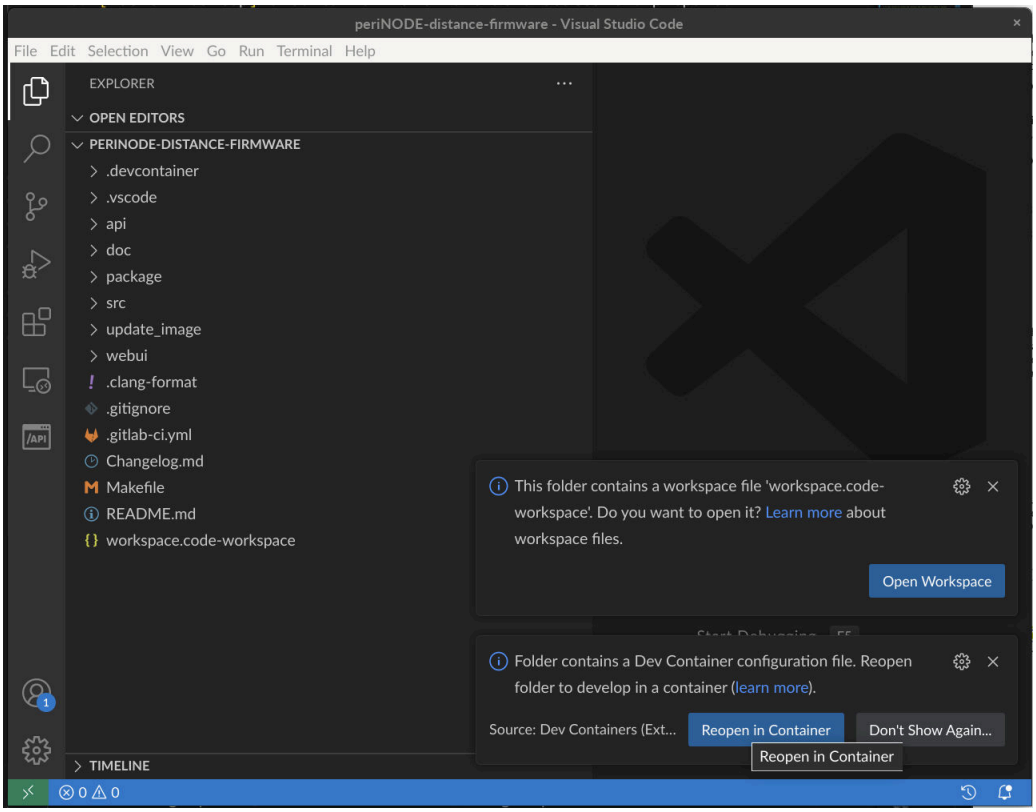
3.2.4 Open Cloned Repository with VsCode



3.2.5 Trust Authors

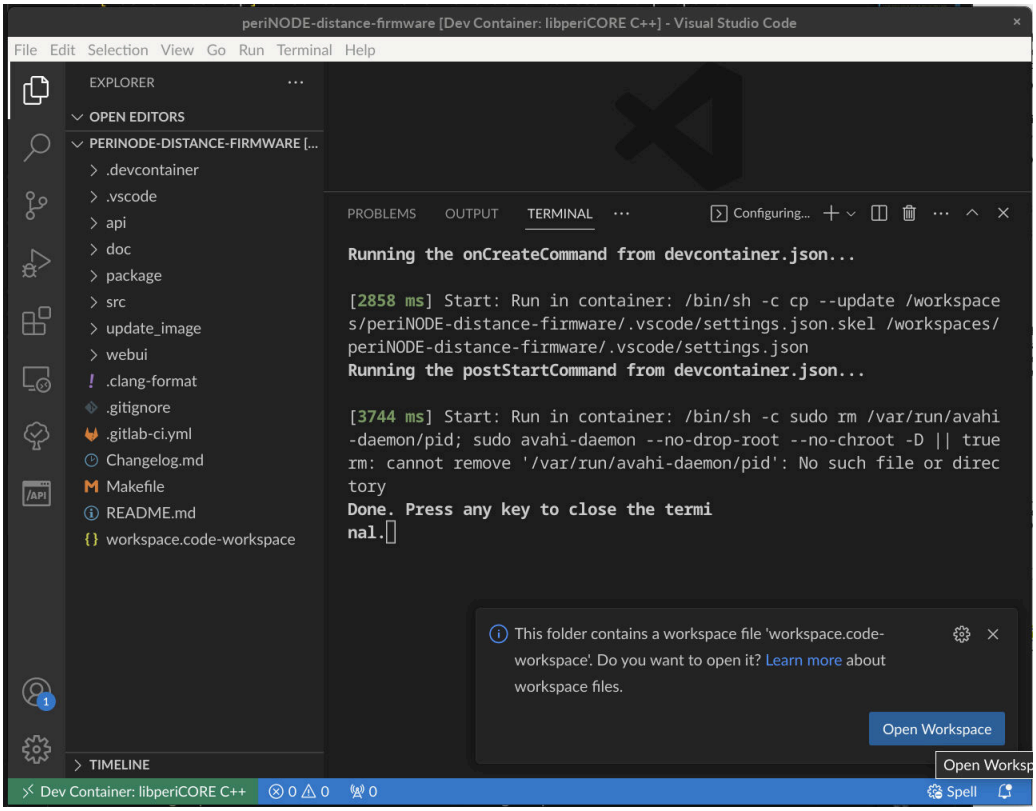


3.2.6 Reopen Repository inside Development Container

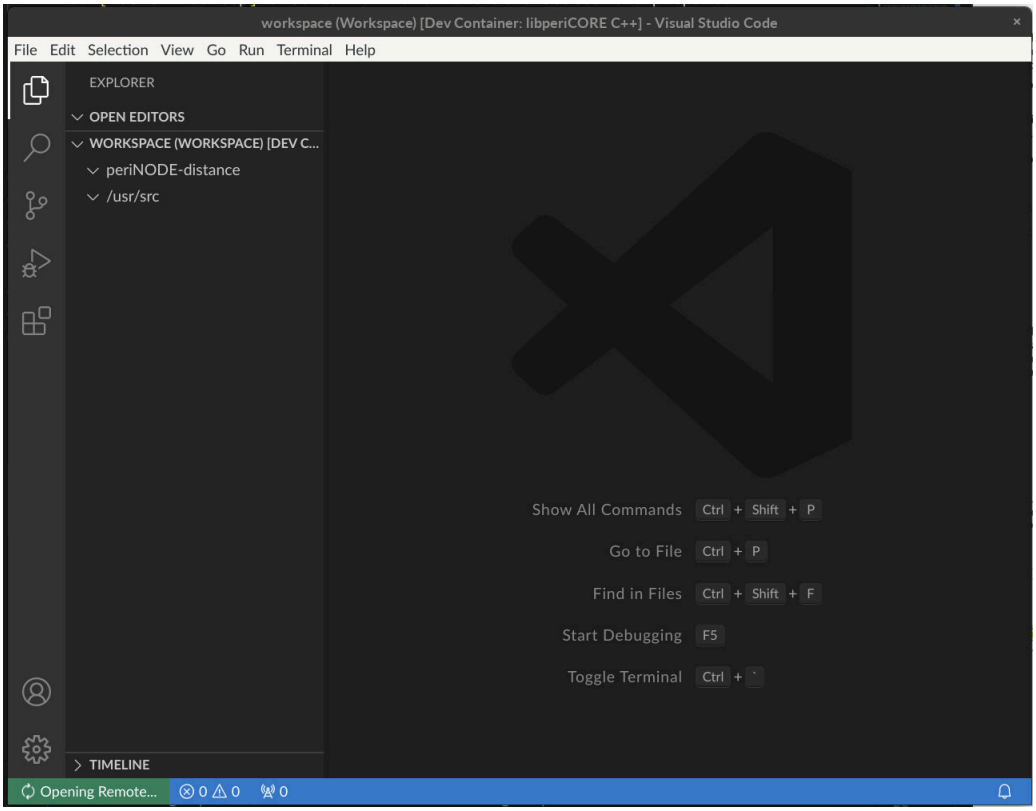


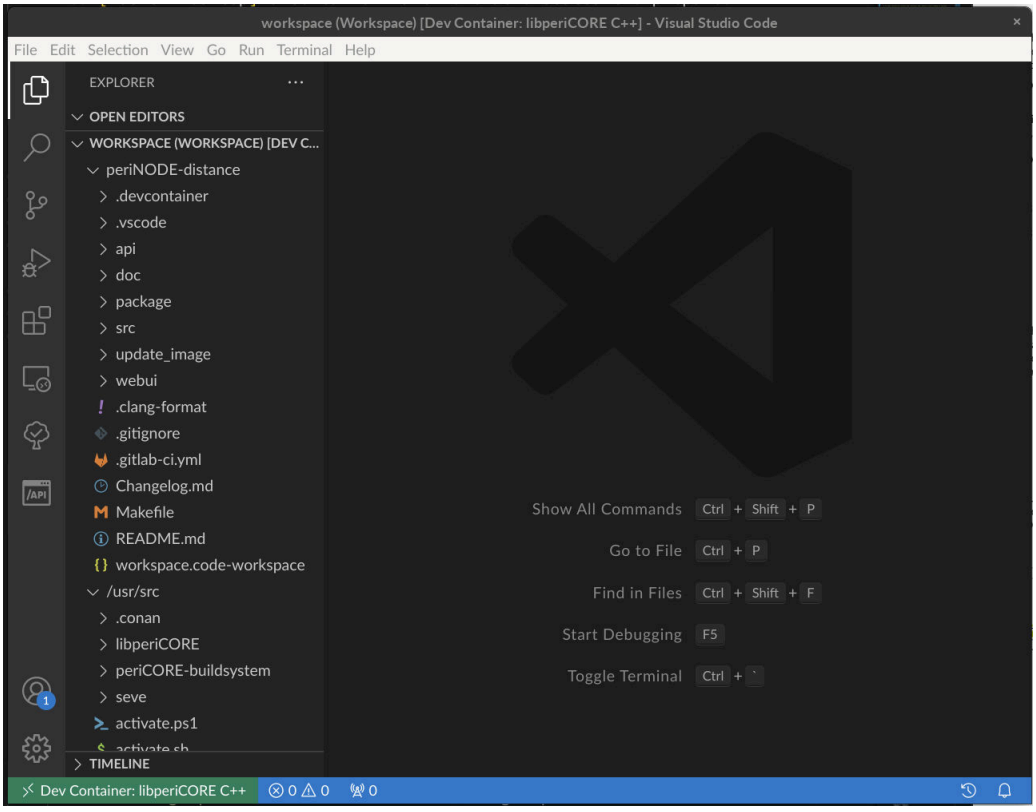
Note: This step results in downloading the development container, which might take a while.

3.2.7 Open Workspace



3.2.8 Opened periNODE-distance Firmware





3.3 Quick Compilation

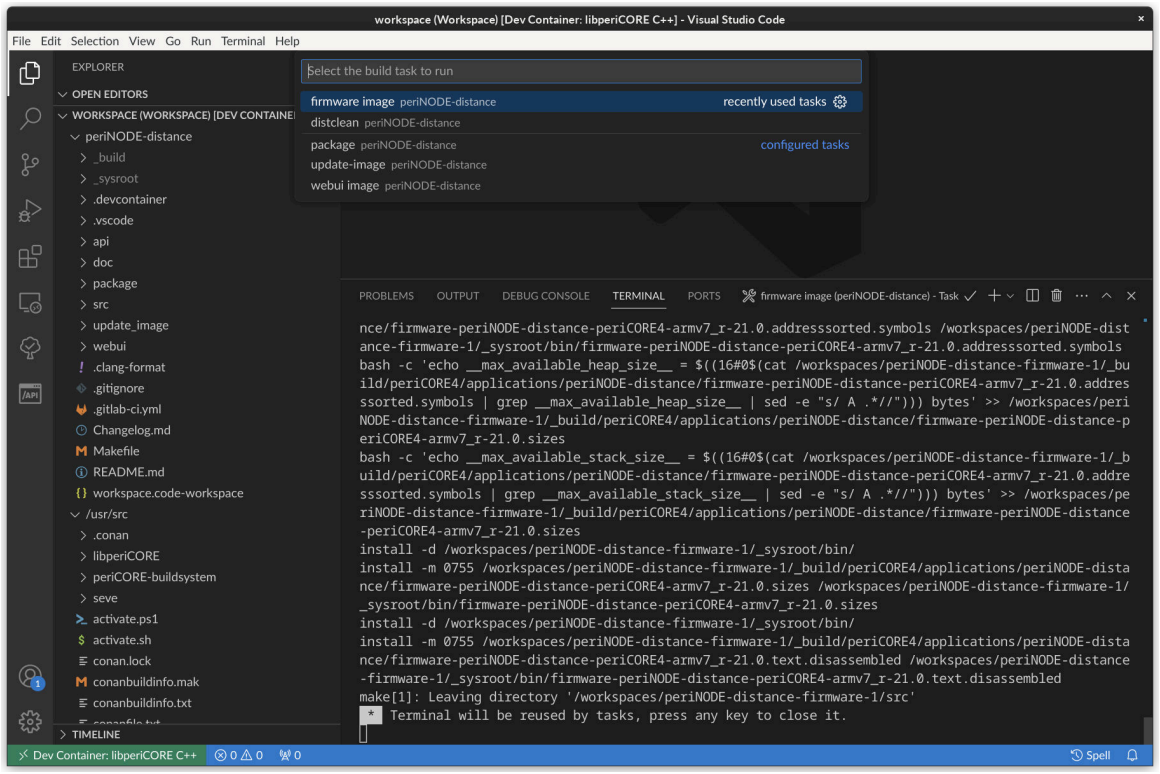


Figure 5: Tasklist of the periNODE-distance firmware environment

Inside the Devcontainer libperiCORE, the firmware can be compiled with the task *firmware image* (*Run Build Task* or *Ctrl + Shift + B*). This is visualized in Figure 5

distclean task removes all build-time artefacts, which is by default located beneath **_build/**.

firmware image creates a compiled binary representation of the periCORE based application firmware. It is the necessary input for a debug session.

package task creates a bundled representation of a periCORE based application firmware. In this format the firmware is distributed and can be updated via the webUI. It includes documentation and the update image.

update-image task creates the update image of the periCORE based application firmware. It includes the firmware image and the webUI image. It can be uploaded to a periNODE via the RESTful interface.

webUI image task creates a download image of the webUI webpage.

3.4 Quick Debug Set Up

After creating the periNODE-distance firmware binary artefacts, the deployment is done automatically during the debug session. The firmware is uploaded at the start of the debug session and is loaded into the RAM of periCORE communication module. The changes are therefore

not persistently stored, what allows a simple recovering of the periCORE Development Kit. Recovery can be performed by restarting the debug session, perform a hardware reset of the periCORE (reset button on ther periCORE Development Board) or perform a power cycle of the periCORE Development Board.

3.4.1 Configure the Debug Target

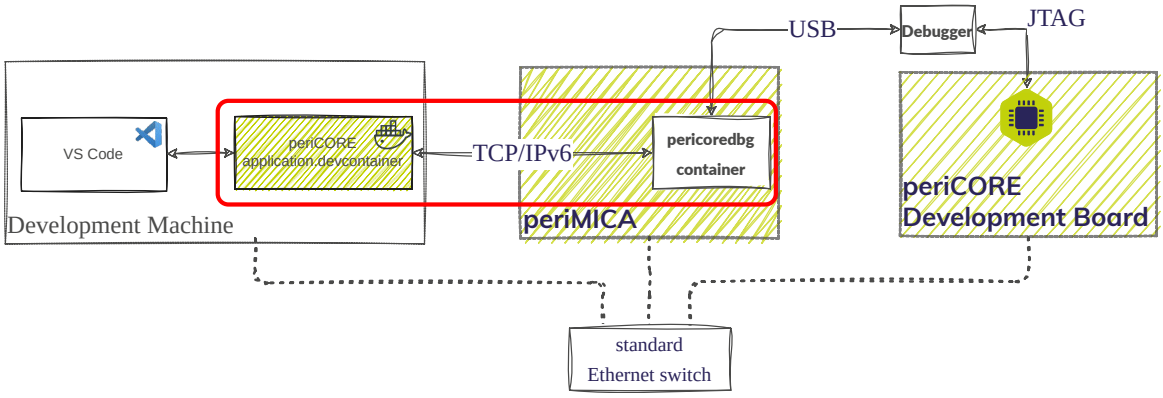


Figure 6: periCORE Development Kit Overview (Debug set up)

The configuration of the debug target is dependent on the used platform (Linux, Windows or macOS) of the Development Machine. A debug session is executed remotely over a network based setup. The communication is initiated from the Development Machine and established between the OCI based libperiCORE Devcontainer [4] and the periCORE debug server periMICA-container [7], as emphasized in Figure 6.

Due to implementation differences of the OCI compatible container runtime, the supported platforms are behaving differently. Therefore, different configurations for the supported platforms are necessary to establish a connection between the Development Machine and the periCORE Development Board.

Configure Debug Target - Linux

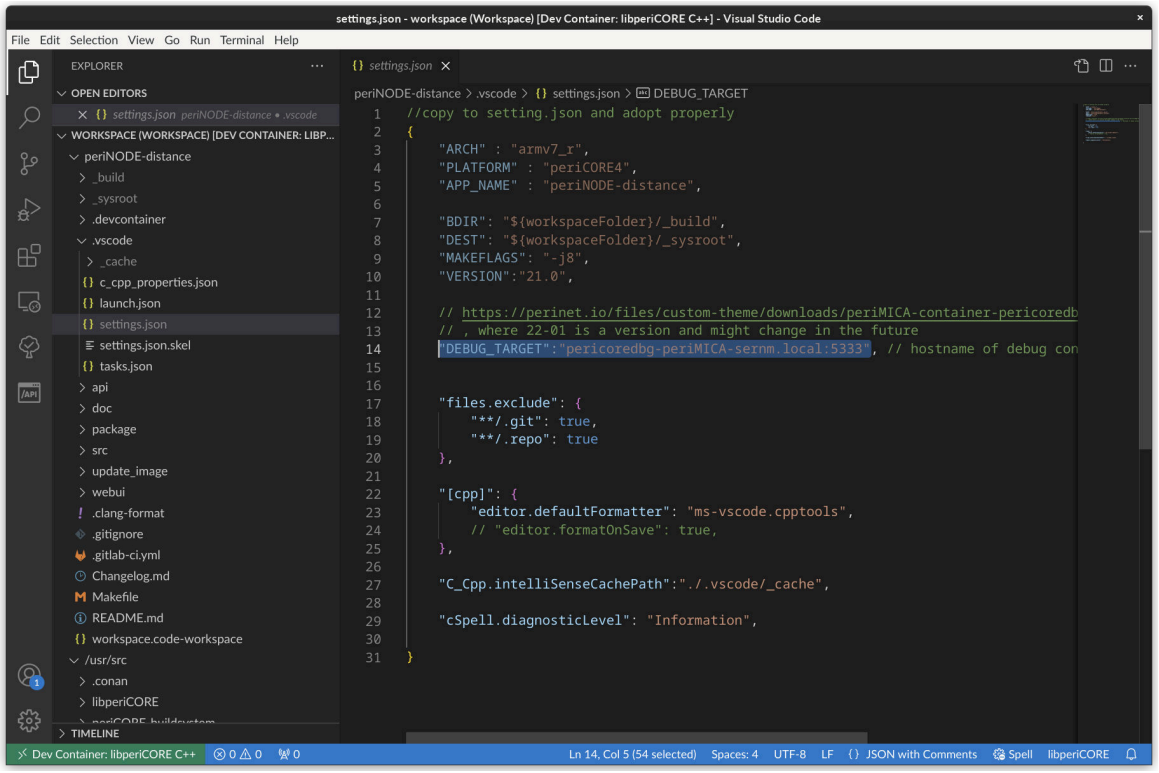


Figure 7: Debug Target configuration of the settings.json file for Linux

The debug target must reference the hostname of the periMICA debug container. In Figure 7 the configured debug target is `pericoredbg-periMICA-sernm.local:5333`, where `pericoredbg-periMIC` is the hostname with the serial number *sernm*, `.local` the local domain name and `:5333` is the port, the debug server is listening on.

Configure Debug Target - Windows and Mac

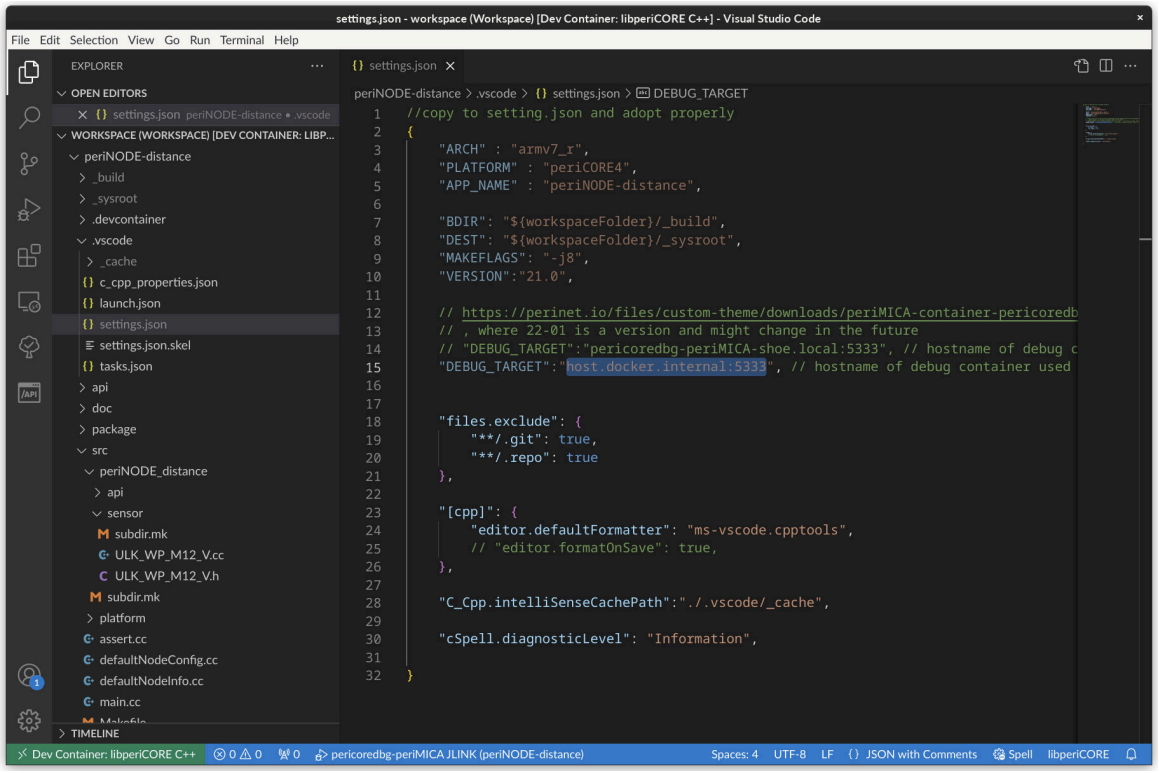


Figure 8: Debug Target configuration of the settings.json file for Windows and macOS

The debug target must reference to `host.docker.internal:5333`, which is then delegated to the hostname of the periMICA debug container on the Development Machine. In Figure 8 the configured debug target is set accordingly.

Note: The real target to the periMICA debug container is set with the invocation of proxy.

Start Proxy for *macOS*

```
socat TCP4-LISTEN:5333,fork,reuseaddr \
    TCP6:pericoredbg-periMICA-sernm.local:5333
```

Listing 1: Socat forwards IPv4 to IPv6 port.(see Section 5.8)

Note: `pericoredbg-periMICA-sernm.local:5333` needs to be adapted accordingly.

Start Proxy for *Windows*

```
netsh.exe interface portproxy add v4tov6 listenport=5333 \
    connectaddress=pericoredbg-periMICA-sernm.local connectport=5333
```

Listing 2: Netsh IPv4 to IPv6 port, (see Section 5.9)

Note: `pericoredbg-periMICA-sernm.local:5333` needs to be adapted accordingly.

3.4.2 Start Debug Session

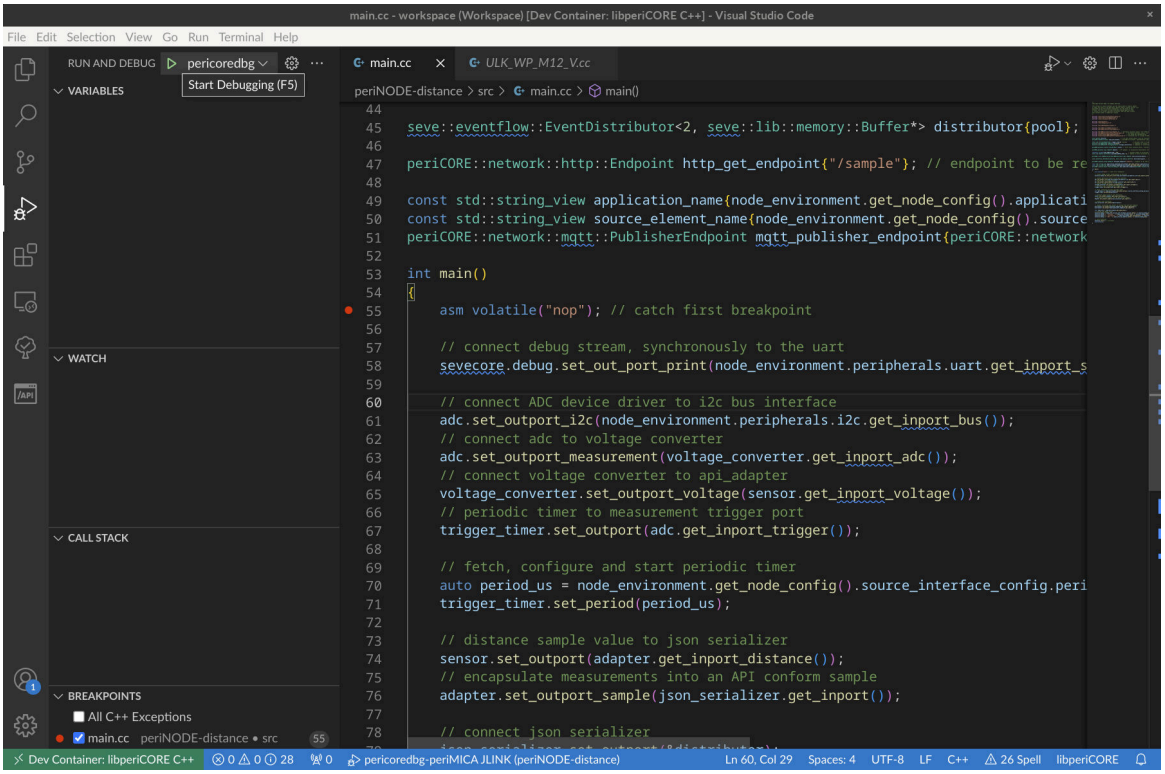


Figure 9: Start a Debug Session with *F5*

A debug session can be started with the key *F5* or as shown in Figure 9 by selecting *pericoredbg - periMICA JLink* debug target and hit the play button.

3.4.3 Break Point

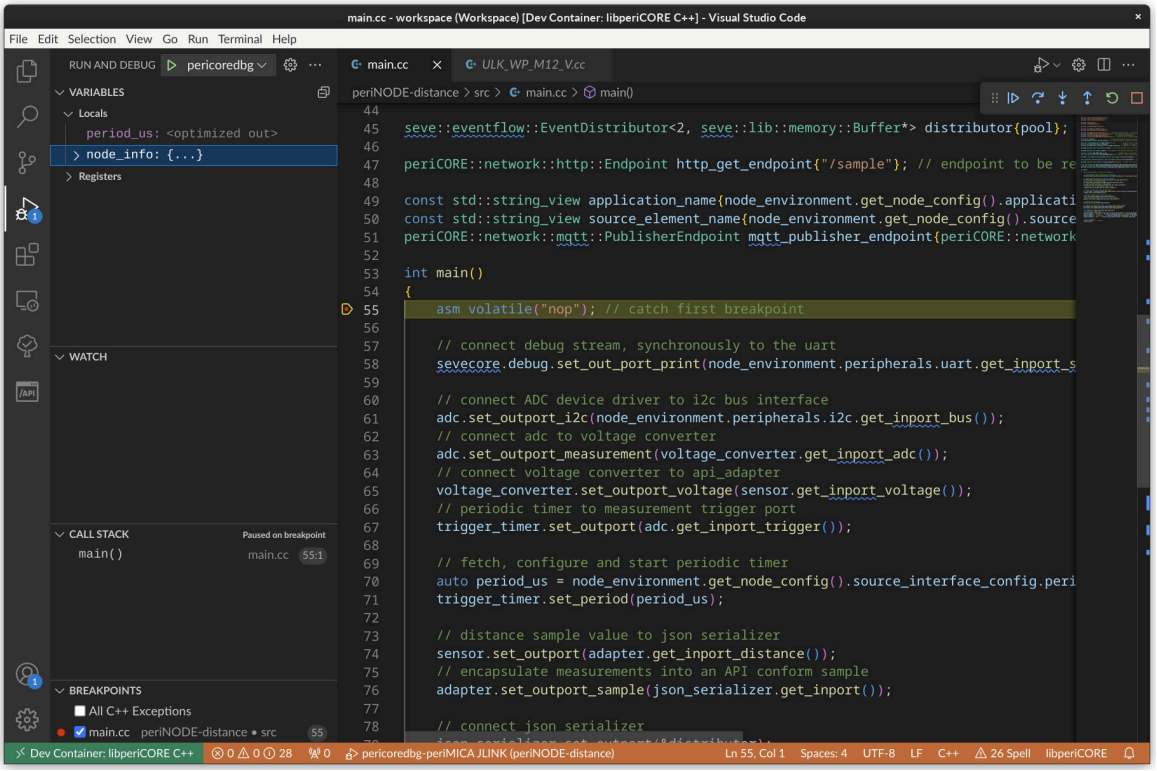


Figure 10: Debug Target configuration of the settings.json file

By default, no breakpoint has been configured which means the debugger will not halt. A breakpoint can be selected directly inside the source code editor and is signaled via a small red dot on the left. Figure 10 shows the debugger stopped at the first line in the main function.

4 Components Assembly

4.1 Step 1: Set up periSTART standard and periSTART Power Supply

We use the *periSTART standard* and its power supply to power the *periCORE Development Board*. *periSTART standard* is a media converter, i.e. it converts between Fast Ethernet (100BASE-TX) and Single Pair Ethernet (100BASE-T1) standards. Please follow the steps below:

1. Connect the *periSTART standard* to the *periLINE hybrid SPE cable*
2. Connect the M8 A-coded cable to the *periSTART standard*
3. Connect the *periSTART power supply* to the *periSTART standard* like in Figure 11:

Figure 11: periSTART standard connected to periLINE and M8 cable

4. Connect the RJ45 plug to a network switch port
5. Connect the *periSTART power supply* to a power outlet

4.2 Step 2: Setup periCORE Development Board

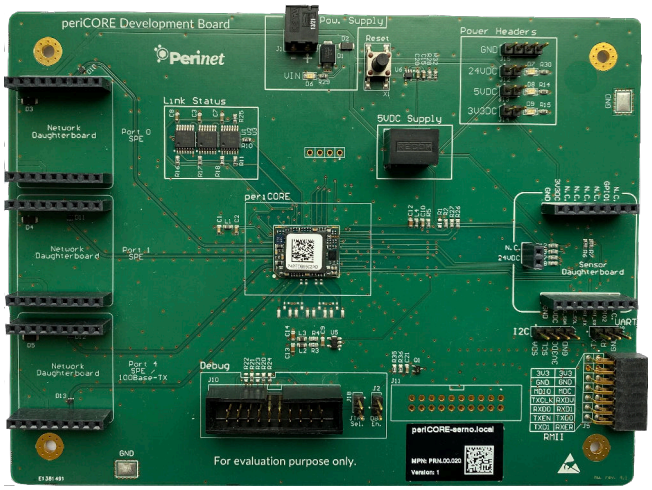


Figure 12: periCORE Development Board

4.2.1 Supply and Network Connection with periSTART

periCORE Development Board needs to be supplied with 24VDC. The power will come via the SPE hybrid network provided by the *periSTART power supply* prepared in the previous step. To supply the board and also have network connection, connect the *M8H Male Daughterboard* to *Port 0 SPE* on the *periCORE Development Board* like in the Figure 13:

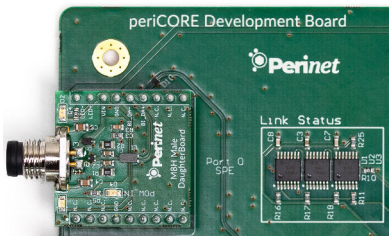


Figure 13: periCORE Development Board with M8H Daughterboard

Now connect the *periLINE* to the M8H Daughterboard like shown in the Figure 14 below:

Figure 14: periCORE Development Board with M8H Daughterboard

4.2.2 Connect the JTAG Interface

Connect the target interface to the JTAG interface (J10) of the periCORE Development Board:

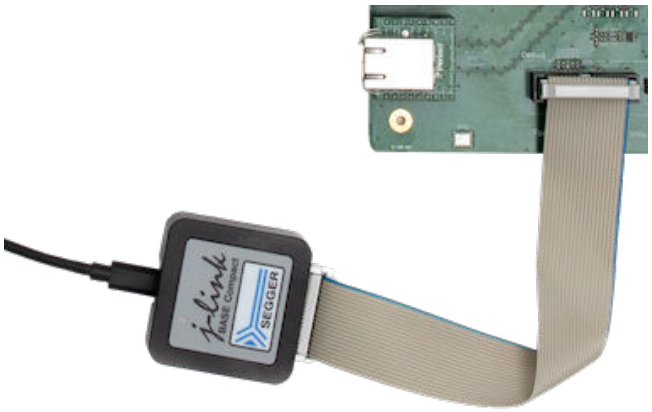


Figure 15: periCORE Development Board - JTAG connection

4.2.3 Connect the Distance Sensor

Connect the cable *M12 Sensor cable* to the Daughterbord 0-10V and the *Distance Sensor* to the other side. Then connect the 0-10V Daughterboard to the *Sensor Daughterboard* slot on the *periCORE Development Board*.

Figure 16: Distance sensor connected to the Daughterboard 0-10V

Note: The sensor is delivered calibrated and ready for usage. The picture of the device can differs depending on the version of the kit.

The following table shows the pin connections for the *M12 Sensor cable* and the Daughterbord 0-10V:

Daughterboard	M12 Sensor Cable
GND	blue
IN	white
TCH	black
24V	brown

Table 2: 010V Daughterboard Cable Connections

4.3 Step 3: Connect the periMICA

4.3.1 Connect the PoE Injector

Connect the PoE injector to the network switch and the power supply to a power outlet (100V..240V).

The PoE injector powers the periMICA and provides network connectivity in the same time. Use the M12-X coded cable to connect periMICA and the PoE injector:

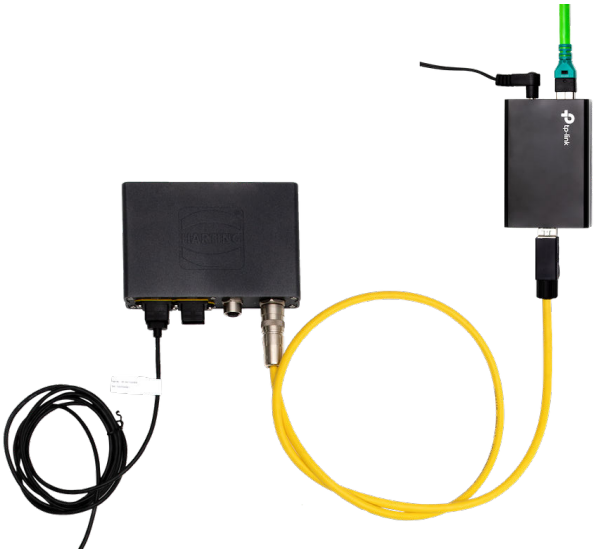


Figure 17: periMICA Powered over Ethernet

4.3.2 Connect the USB JTAG to the periMICA

Connect periMICA to the *J-Link debugger* attached to the *periCORE Development Board*, using the one of the USB port available like in the Figure 18:

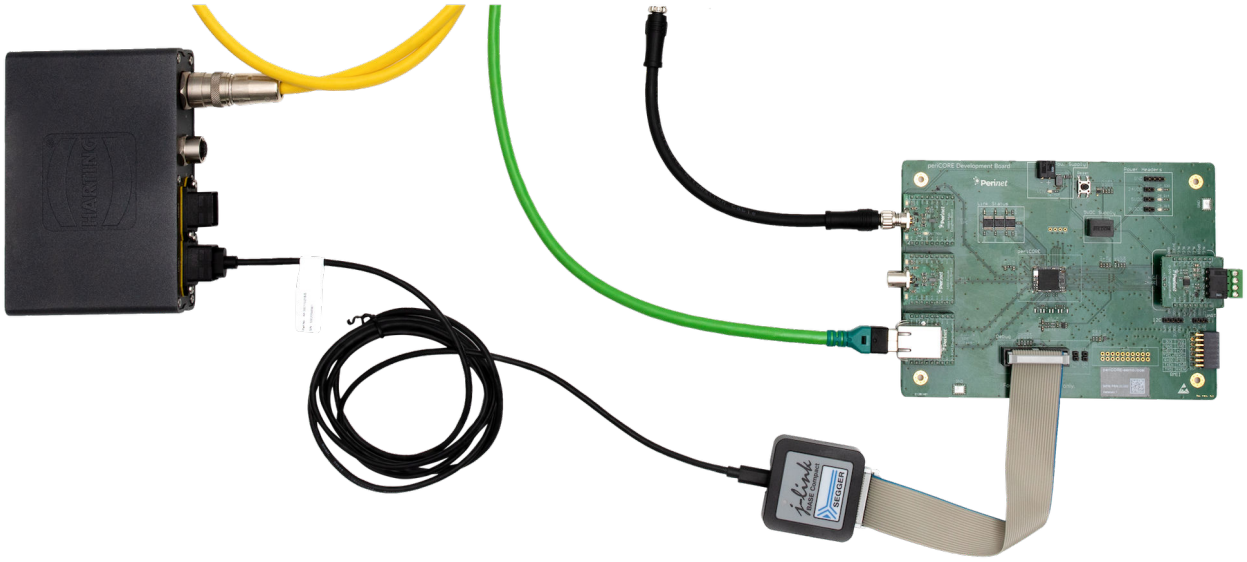


Figure 18: periMICA connected to JTAG

Make sure to connect the JTAG before starting the *pericoredbg container*, otherwise a reboot might be necessary.

The complete setup, including the *periMICA*, looks as in Figure 19:

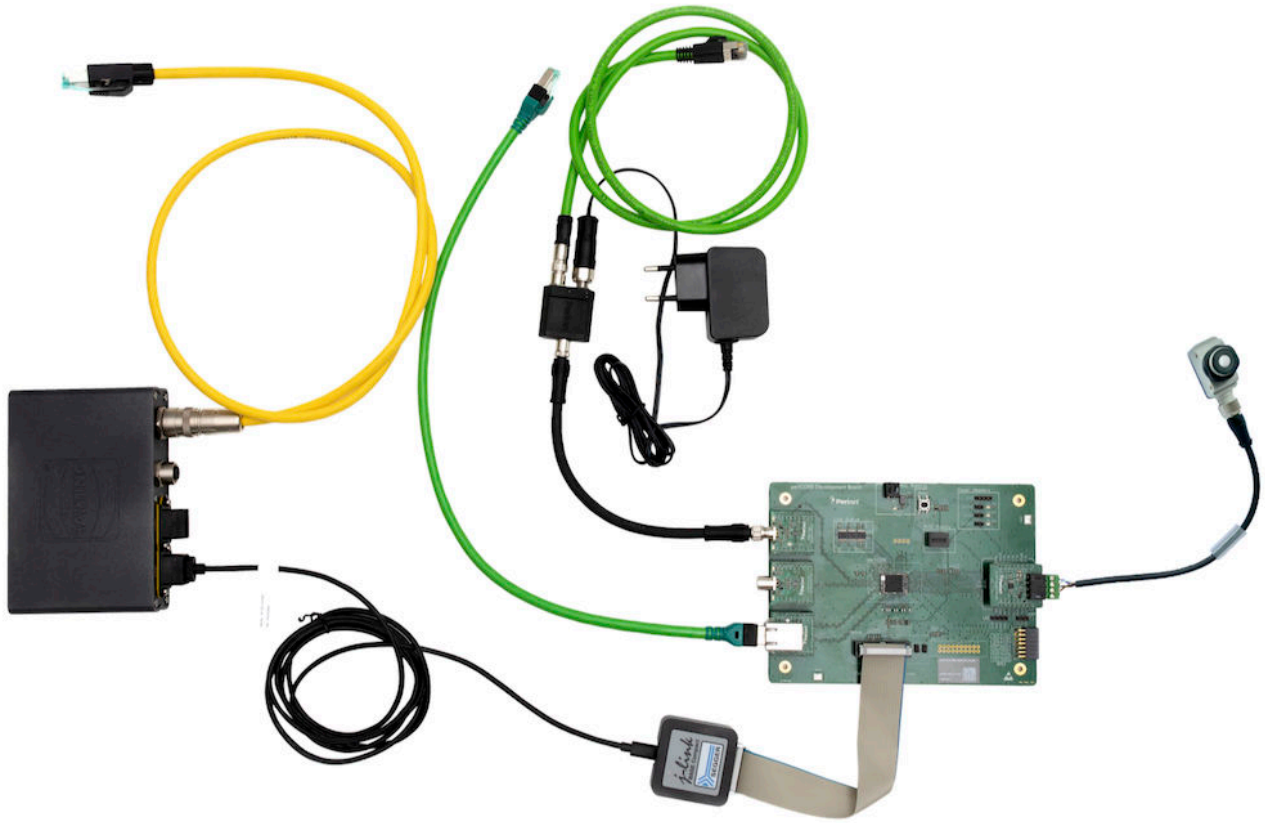


Figure 19: periCORE Development Board - setup complete

In this section we finished the hardware setup of the *periCORE Development Kit*. The *periCORE Development Board* with the sensor attached is powered and connected to the network and to the J-Link debugger. *periMICA* is powered and connected to the J-Link as well.

You can also take advantage of the others Daughterboards included in the kit to supply the board, to provide the network connection or use another sensor application when you need. Here we provide just one example of a complete setup that is functional with the goal to have the application distance working at the end. For further information please refer to the periCORE Development Kit User guide [8].

We are now ready to proceed with the setup of the software environment.

5 Software Environment Setup

5.1 Install and Configure Docker

The *Devcontainer libperiCORE* [4] is a *docker* container based on Open Container Initiative (OCI). It delivers the ARM toolchain, compiler, debugger, libraries and others support development tools needed for developing applications for your *periCORE SPE communication module*.

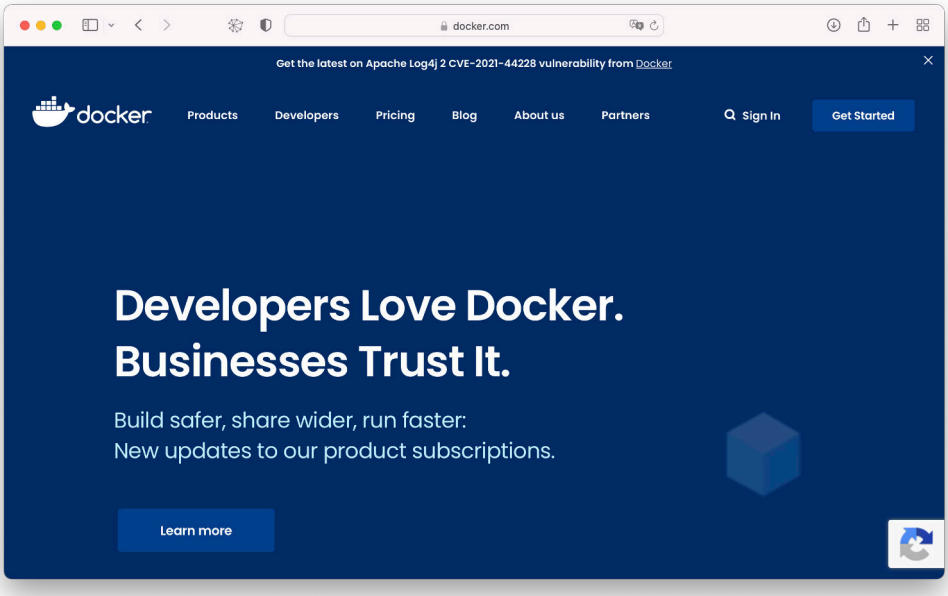


Figure 20: Docker web site

The *libperiCORE* is a library embedded in the *Devcontainer libperiCORE* that provides the build dependencies and Makefiles prepared to use the toolchain. Please refer to periCORE Datasheet [6] for more detailed information about the *libperiCORE* library.

5.1.1 Install Docker Desktop for Windows or macOS

The Docker Desktop application is the tool recommended for easily installing everything you need to work with Docker containers. It is available for Windows and macOS environments.

For more information on how to install the Docker Desktop please refer to:

- **macOS**: https://docs.docker.com/desktop/mac/install/
- **Windows**: https://docs.docker.com/desktop/windows/install/

5.1.2 Install Docker Engine for Linux

The Docker for Linux systems is called Docker Engine and supports different Linux distributions. For simplicity, we provide an installation example for Debian/Ubuntu:

1. Setup the repository

```
$ sudo apt-get update
$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Listing 3: Prepare repository and install dependencies

2. Add Docker's official GPG key

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /
usr/share/keyrings/docker-archive-keyring.gpg
```

Listing 4: Add Docker GPG key

3. Setup the **stable** repository

```
$ echo \
     "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/debian \
      (lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /
dev/null
```

Listing 5: Setup the stable repository

4. Install the Docker Engine

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Listing 6: Install Docker engine

5.1.3 Configure the Docker Engine on Linux

The default configuration of the docker daemon does not run IPv6, therefore we need to configure it by editing `/etc/docker/daemon.json` and adding the following lines:

```
{
    "ipv6":true,
    "fixed-cidr-v6": "2001:db8:1::/64"
}
```

Listing 7: Parameters needed on daemon.json

5.2 Visual Studio Code

Visual Studio Code is a source-code editor that can be used with a variety of programming languages and has an extended support for the C++ programming language. VS Code can be extended with *Extensions* in order to make the development more comfortable.

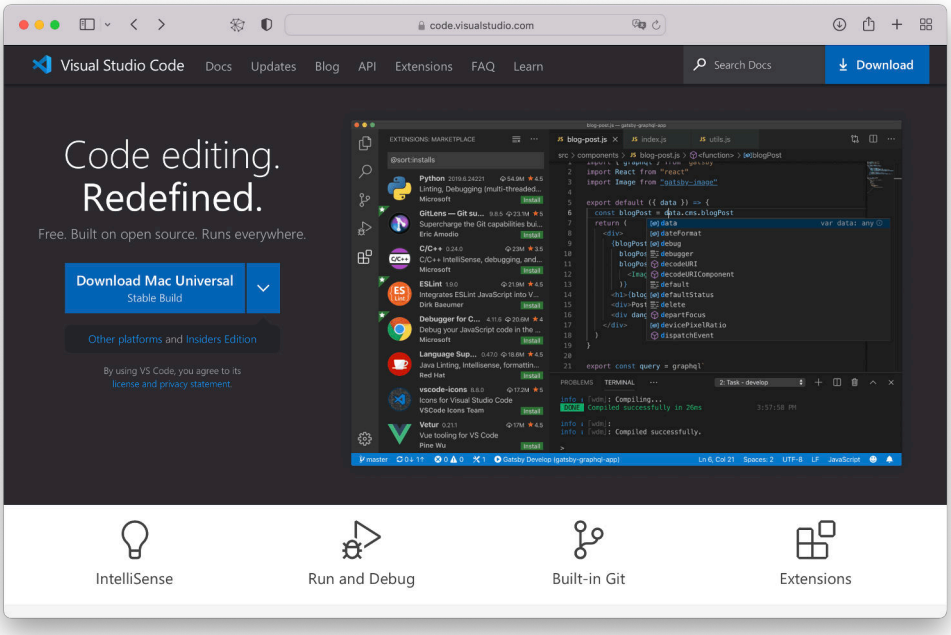


Figure 21: VS Code web site

Download VS Code from https://code.visualstudio.com/Download, select your operating system and proceed with the installation.

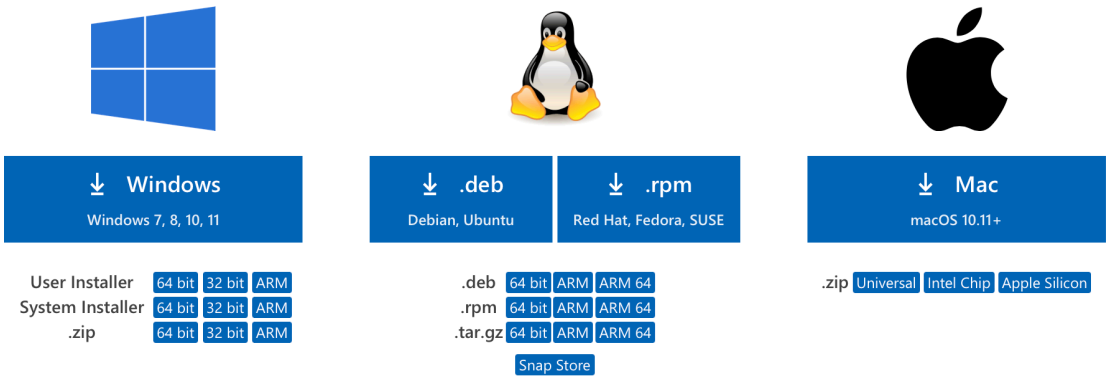


Figure 22: VS Code download platforms

5.2.1 Basic VS Code Configuration

Open VS Code and press **Shift + Ctrl + P** (or **Shift + Command + P** for macOS) and type ”shell command” in the prompt (Figure 23). Select the option to include ***code*** in the path and close VS Code to apply the configuration.

Figure 23: Install VS Code in PATH

Note: The referred *PATH* is an environment variable on operating systems that specifies a set of directories where executable files are located.

There is no ”shell command” option in Windows systems to install `code` command in PATH. Instead, the user receives the offer to include it during the installation process. In case you skipped this step, please refer to section Section 5.9.

From now on, VS Code can be optionally opened through the command line simply by typing:

```
code .
```

Listing 8: Open VS Code

5.2.2 Install Dev Containers Extension

On the left side of the VS Code window, we can see the symbol for the Extensions (Figure 24):

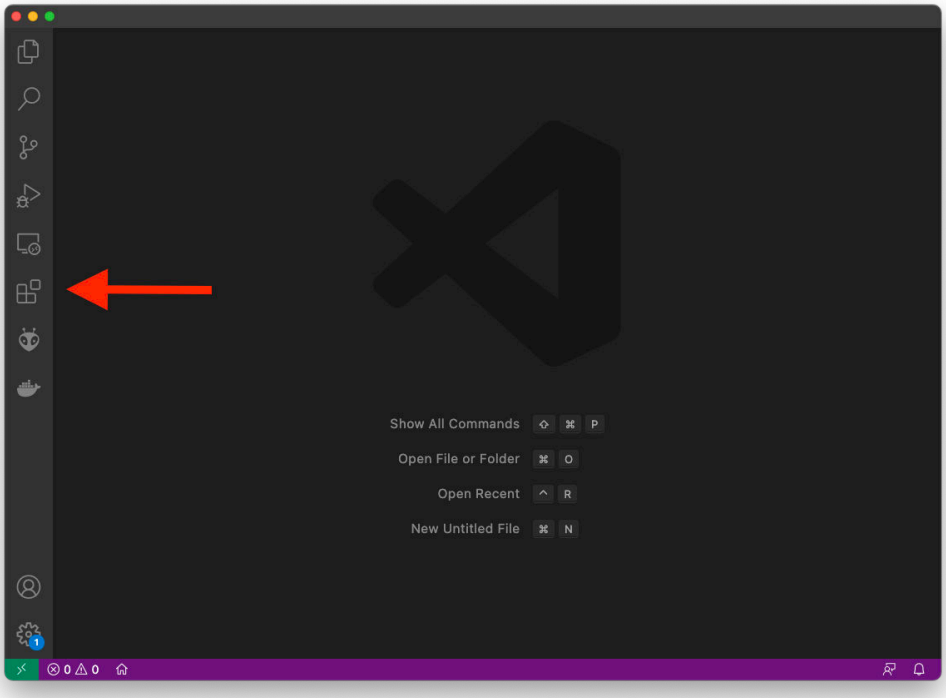


Figure 24: VS Code Extensions

In VS Code **Extensions**, search for **Dev Containers** and install it:

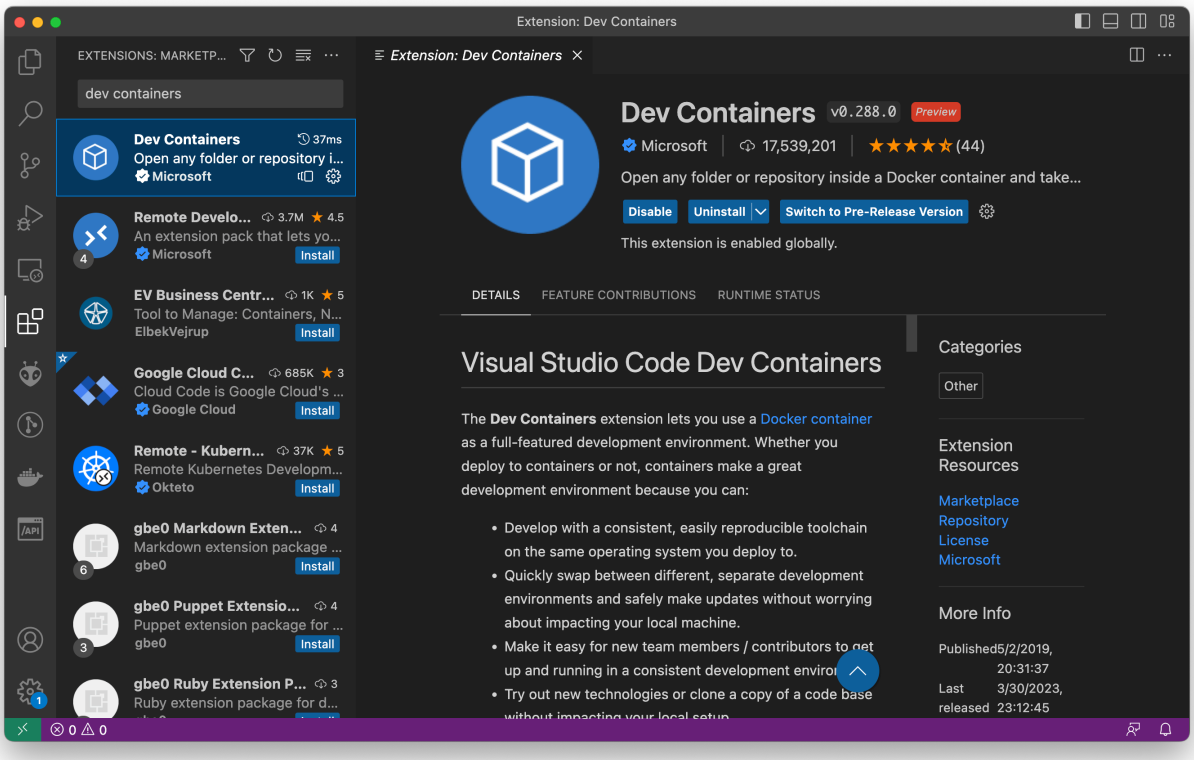


Figure 25: VS Code Remote Container extension

5.3 Download and Install the *pericoredbg* periMICA Container

The *pericoredbg* periMICA container is useful for debugging your application firmware for the *periCORE SPE communication module*. It provides remote access to a JTAG-debugger with VS Code. The periCORE Development Kit includes remote debugging capabilities.

Note: The periCORE Development Kit is delivered with preinstalled *pericoredbg periMICA* container.

5.3.1 Download pericoredbg periMICA Container

Download the *pericoredbg* periMICA container from https://perinet.io/downloads.

5.3.2 Install the pericoredbg container on the periMICA

Open a browser and access the periMICA homepage e.g. https://perimica-serno.local. The name of the modular edge computer, as well as the login and password, can be found on the periMICA label:

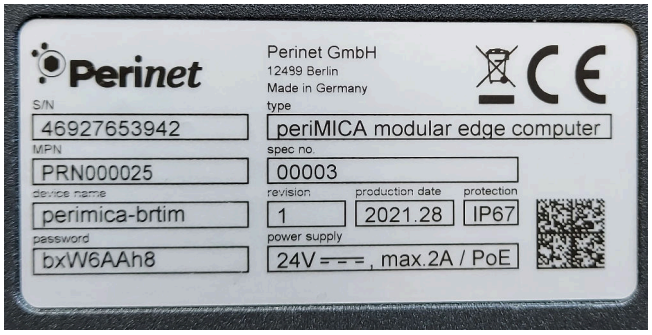


Figure 26: periMICA label

Type the *periMICA*.local name in the URL field of your preferred browser and the login screen will be displayed (Figure 27):

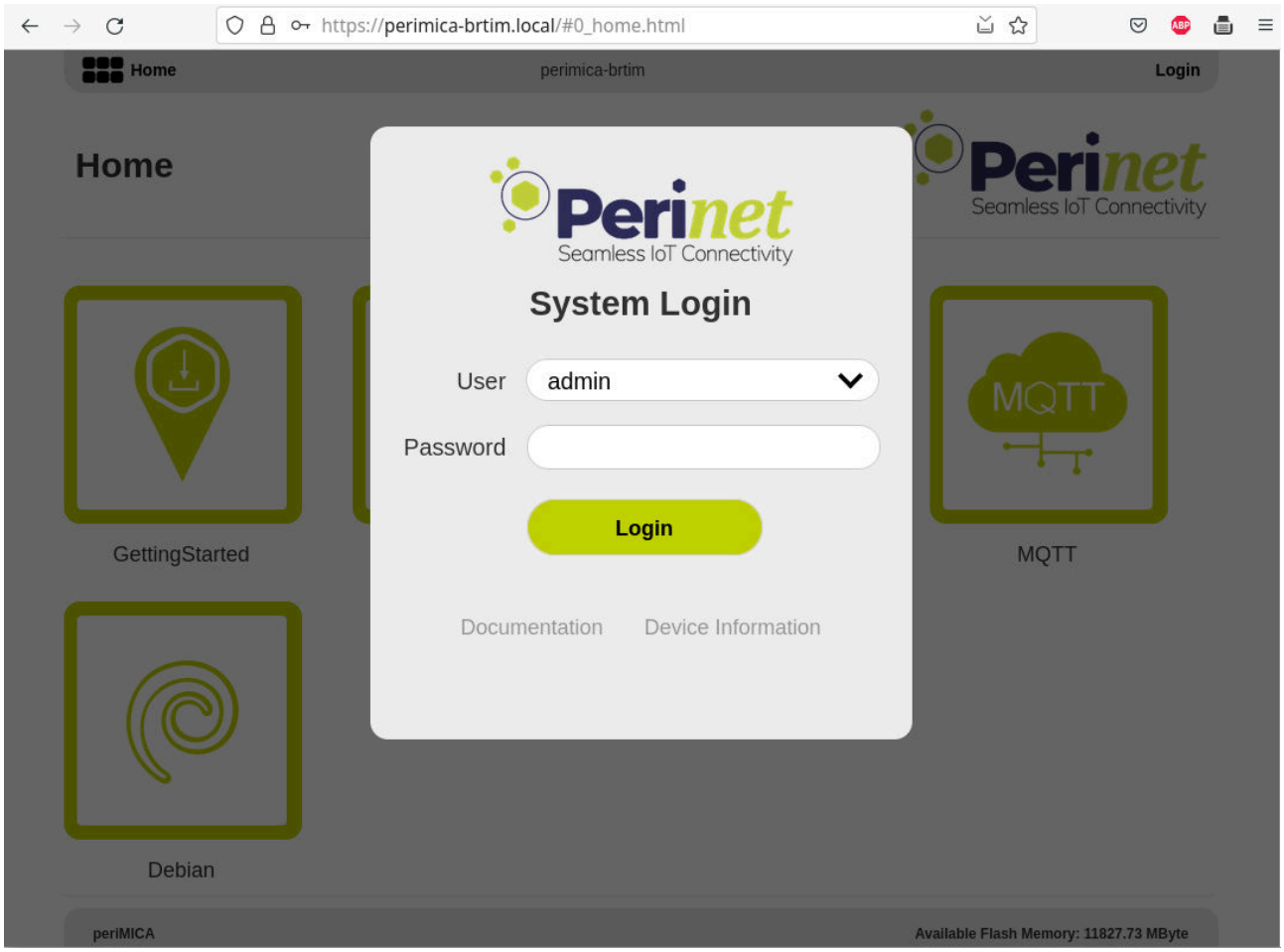


Figure 27: periMICA Login

After logging in, click on **Install**:

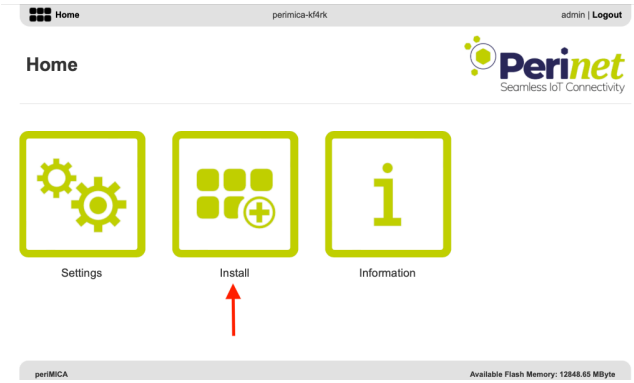


Figure 28: periMICA Install

Type the name of the new container. For simplicity, we suggest using *pericoredbg*, as it appears throughout this guide. Click on **Continue** to proceed:

Figure 29: periMICA install **pericoredbg** container

After the installation, please start the container by right-clicking on the new icon *pericoredbg* and on **Start**:

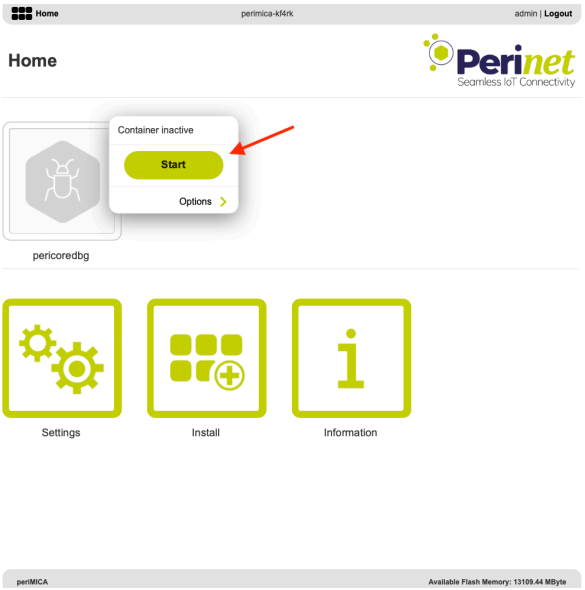


Figure 30: Start pericoredbg container

Through *pericoredbg* container, the periCORE module can be debugged or programmed using

the USB A port on the periMICA and the J-Link debugger connected to the *periCORE Development Board* through the network connection between VS Code and GDB Server.

In the *pericoredbg* webUI is possible to observe the output messages of the debug server like shown in the figure 31.

Figure 31: Start pericoredbg container

5.4 Get the devcontainer Docker Image

Perinet provides the `Devcontainer libperiCORE`. It contains all tools needed for developing periCORE based firmware applications. There is no need to pull this container explicitly. The `.devcontainer` environment which is part of the *periNODE-distance firmware* example application keeps control of the devcontainer. When opening the application dedicated workspace (*workspace.code-workspace*) for the first time the devcontainer gets pulled in the right version implicitly.

5.5 Get the periNODE-distance Firmware Source Code

The *periNODE-distance firmware* [5] source code can be downloaded as a **.zip** file or cloned with `git`:

```
$ git clone https://gitlab.com/perinet/periNODE-0-10V/periNODE-distance-firmware.git
```

Listing 9: Clone the repository

Download the **.zip** file from periNODE-distance firmware repository[3] and unzip the `periNODE-distance-f`

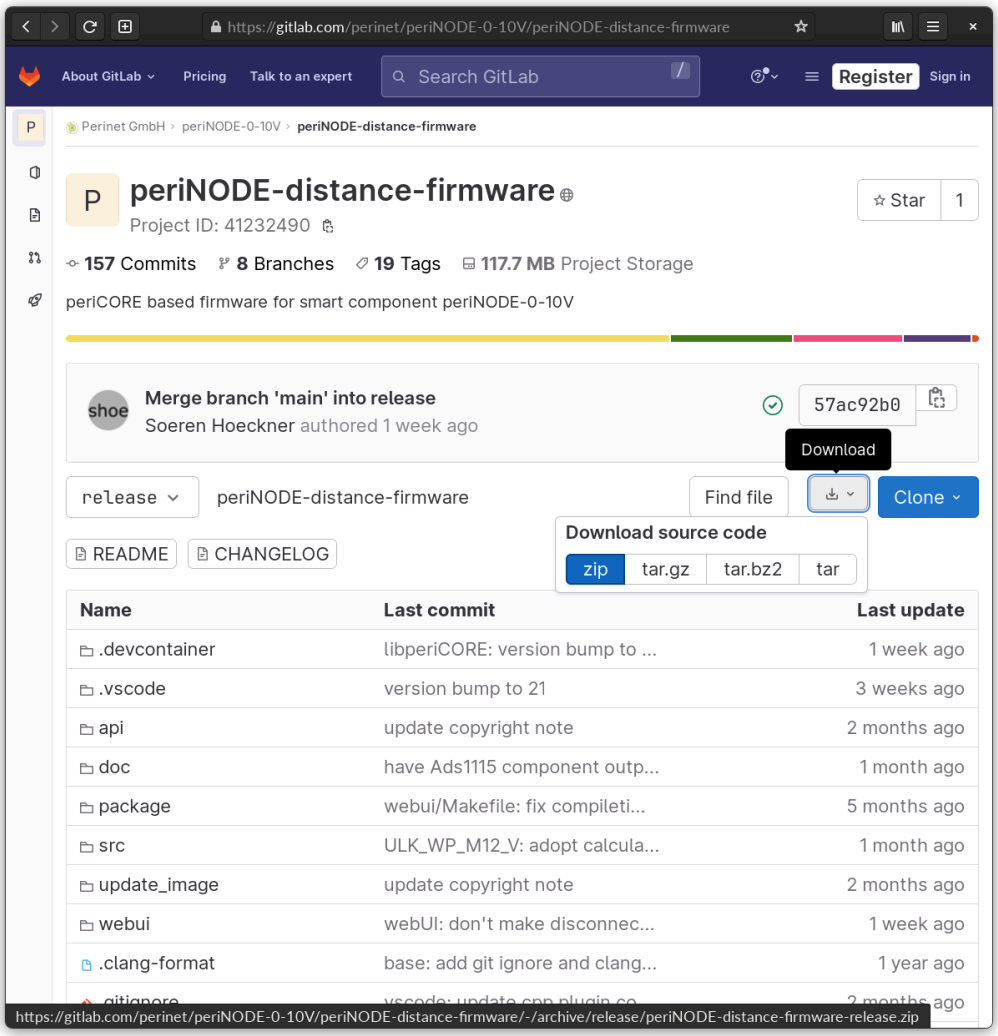


Figure 32: Obtain the periNODE-distance Sources

5.6 Open the periNODE-distance workspace

Open a terminal, go to the folder with the `periNODE-distance` source code and open the workspace:

[3] https://gitlab.com/perinet/periNODE-0-10V/periNODE-distance-firmware.git

```
$ cd periNODE-distance-firmware
$ code workspace.code-workspace
```

Listing 10: Open the workspace

At the first time trying to open the workspace the following security message is expected:

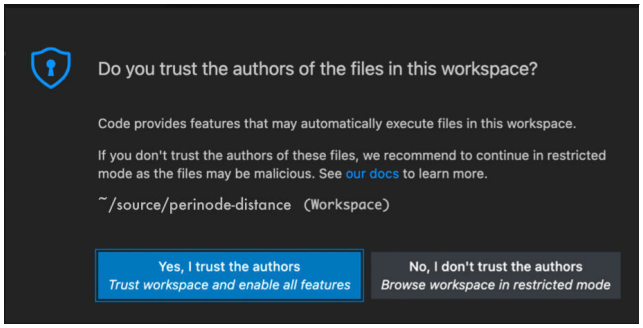


Figure 33: VS Code Trust Security

This means the vscode will run automatically pre-defined tasks, debug the application and will apply the settings in the environment as well. You can trust on *Perinet* source code clicking on *Yes, I trust the authors*. Optionally you can choose don't trust and enter in a *Restrict Mode* which will pop-up a confirmation message, as shown in Figure 35, when you ask to run the tasks, or debug the code. Please refer to VS Code Documentation [17] to get more information about VS Code Workspace Trust.

VS Code will display an option that offers to **Reopen in Container**:

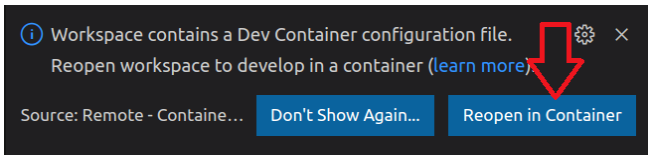


Figure 34: Open periNODE-distance environment within docker container

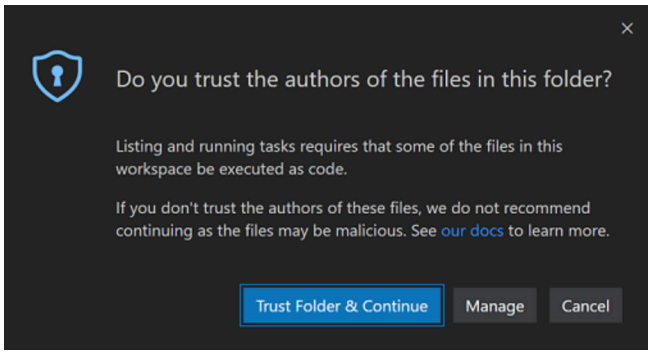


Figure 35: Open periNODE-distance environment within docker container

After VS Code reopens in the container, we see the *periNODE-distance* structure (Figure 36):

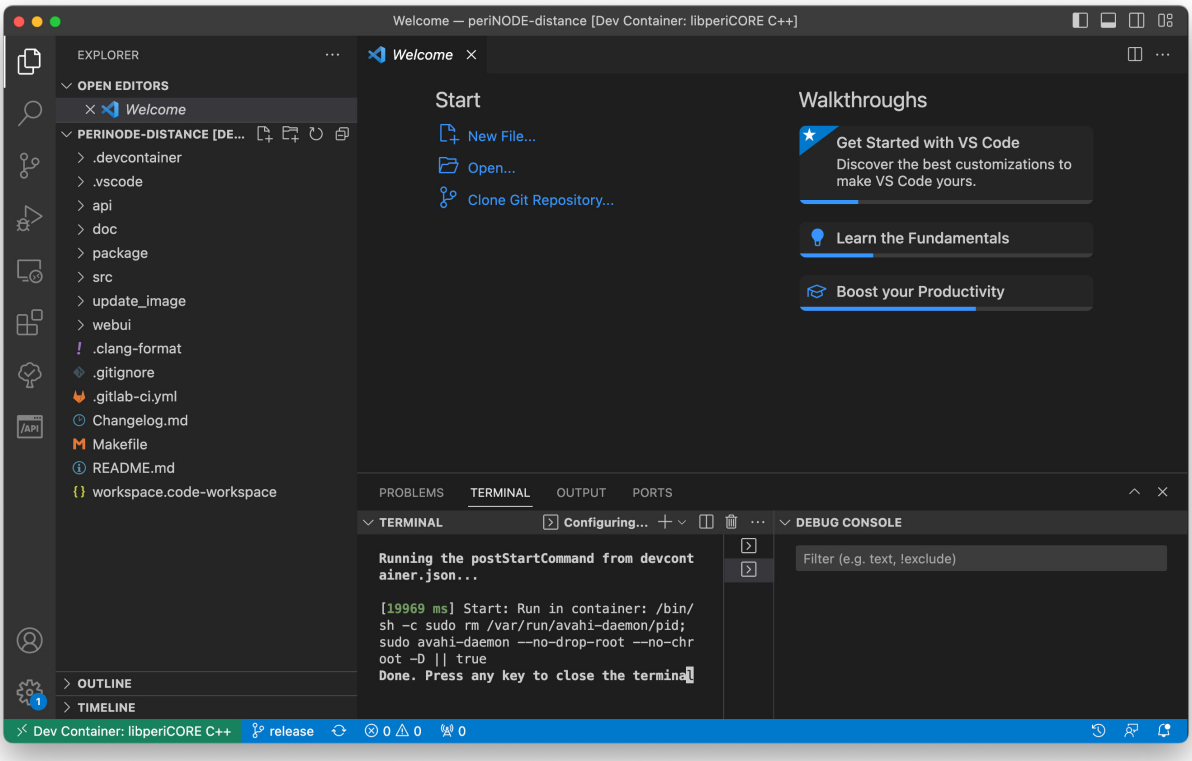


Figure 36: VS Code Running in Container

5.7 Configure the periNODE-distance

5.7.1 Configure settings.json File

On the left side, open the folder `.vscode` and click on the file: `settings.json` (see Listing 11).

```
//copy to setting.json and adopt properly
{
  "ARCH" : "armv7_r",
  "PLATFORM" : "periCORE4",
  "APP_NAME" : "periNODE-distance",

  "BDIR": "${workspaceFolder}/_build",
  "DEST": "${workspaceFolder}/_sysroot",
  "MAKEFLAGS": "-j8",
  "VERSION":"20.0",

  // https://perinet.io/files/custom-theme/downloads/periMICA-container-pericoredbg-22-01.tar
  // , where 22-01 is a version and might change in the future
  "DEBUG_TARGET":"pericoredbg-periMICA-sernm.local:5333", // hostname of debug container used for periCORE4 debug env


  "files.exclude": {
     "**/.git": true,
     "**/.repo": true
  },

  "[cpp]": {
     "editor.defaultFormatter": "ms-vscode.cpptools",
     // "editor.formatOnSave": true,
  },

  "C_Cpp.intelliSenseCachePath":"./.vscode/_cache",

  "cSpell.diagnosticLevel": "Information",
}
```

Listing 11: Setup workspace variables - settings.json

Replace the `sernm` in the variable `DEBUG_TARGET` with the periMICA name (written on periMICA label as in Figure 26).

The name of the debug container is composed of the container name selected during installation the default installation is called **pericoredbg-periMICA-name**.local. This is the DNS name of your debug container.

In this file you can also configure the `path` to the auxiliary folders to place the end files created during the firmware building process. The default values can be kept, although they will be created only after the first build process run.

"BDIR": folder to place the intermediate files created during the building process that are needed to create the firmware binary. The default is set to `_build`.

"DEST": folder to place the final firmware binary and the update image file. The default is set to `_sysroot`.

Note: The *VS Code* on Windows systems needs to be restarted in order to apply the changes in settings.json.

5.7.2 Configure tasks.json File

The `tasks.json` file is used to configure the most used tasks during the development. For example, for building the application, the `make` command is extensively used, and its corresponding task can be called by pressing the shortcut **Ctrl + Shift + B** on Windows and Linux. For macOS, use **Command + Shift + B**. Additionally, it can used ***Menu Terminal →Run Task...*** and select the proper option.

Take advantage of the *VS Code* tasks panel by binding the most used tasks to a keyboard shortcut. This can help any automate repetitive task, streamline the development workflow, and improve productivity.

Note: The example application periNODE-distance has already configured the most used tasks and in order to start no actions is expected here.

5.7.3 Configure launch.json File

When you start a debugging session in *VS Code*, it uses the settings in the file `launch.json` to determine which executable to run, which arguments to pass to it, and how to attach the debugger to the running process. It fetches some variables configured in the `settings.json`.

The `launch.json` is already prepared to launch the debugger, and to run the example application no modifications are expected here.

You can also create as many configuration profiles as you desire. In the `launch.json` file we can take advantage of the variables configured in `settings.json`. Please take a look at the example of the default file that comes with the *periNODE-distance* package:

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "[bcm] pericoredbg-periMICA JLINK",
      "type": "cppdbg",
      "request": "launch",
      "args": [],
      "program": "${config:DEST}/bin/firmware-${config:APP_NAME}-${config:PLATFORM}-${config:ARCH}-${config:VERSION}.elf",
      "stopAtEntry": true,
      "cwd": "${workspaceRoot}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerPath": "/usr/local/gcc-arm-none-eabi/bin/arm-none-eabi-gdb",
      // bug workaround, where the remote target cannot be stopped when the debugServer is not started directly by vscode
      "debugServerPath": "/bin/bash",
      "debugServerArgs": "-c \"echo Workaround Started; trap - SIGTERM SIGINT SIGHUP; while true; do sleep 10; done;\"",
      "serverStarted": "Workaround Started",
      // bug workaround end
      "sourceFileMap": {
        "${workspaceRoot}":"${workspaceFolder}"
      },
      "logging": {
        "engineLogging": true,
        "exceptions": true,
        "moduleLoad": true,
        "programOutput": true,
        "trace": true,
        "traceResponse": true
      },
      "hardwareBreakpoints": { "require": true, "limit": 2 },
      "targetArchitecture": "arm",
      "customLaunchSetupCommands": [
        {
          "description": "Enable pretty-printing for gdb.",
          "text": "-enable-pretty-printing",
          "ignoreFailures": false
        },
        {
          "description": "Load executable into debugger.",
          "text": "file ${config:DEST}/bin/firmware-${APP_NAME}$-${config:PLATFORM}-${config:ARCH}-${config:VERSION}.elf",
          "ignoreFailures": false
        },
        {
          "description": "Connect to gdb server.",
          // "text": "-target-select remote ${config:DEBUG_TARGET}:5333",
          "text": "-target-select remote ${input:remote_debug}",
          "ignoreFailures": false
        },
        {
          "description": "reset and halt target",
          "text": "monitor reset; monitor halt",
          "ignoreFailures": false
        },
        {
          "description": "Load executable to target.",
          "text": "-target-download",
          "ignoreFailures": false
        },
        {
          "description": "breakpoint at entry",
          "text": "break main",
          "ignoreFailures": false
        },
        {
          "description": "Stop at assert",
          "text": "break platform__assert",
          "ignoreFailures": false
        }
      ]
    },
  ],
  "inputs": [
    {
      "id": "remote_debug",
      "description": "Enter remote debugcontainer uri",
      "default": "pericoredbg-periMICA-serno.local:5333",
      "type": "promptString",
    }
  ]
}
```

Listing 12: Debug configuration - launch.json

5.8 Specific Configuration for macOS

The docker network developed for macOS is implemented with a virtual machine and does not create a `docker0` interface like on Linux systems. This interface helps the docker container to communicate with the host network directly.

Another disadvantage of macOS docker implementation is observed when the special DNS name resolves only to the IPv4 address in the host. For further information please refer to https://docs.docker.com/desktop/mac/networking/.

Due to these limitations, we need to create a way to communicate with the host machine network and afterwards with *pericoredbg* container.

First of all we need to use the special DNS provided (`host.docker.internal:5333`) in order to communicate with the host machine. Configure it in your `settings.json` file through the variable `DEBUG_TARGET` as showed in Figure 37:

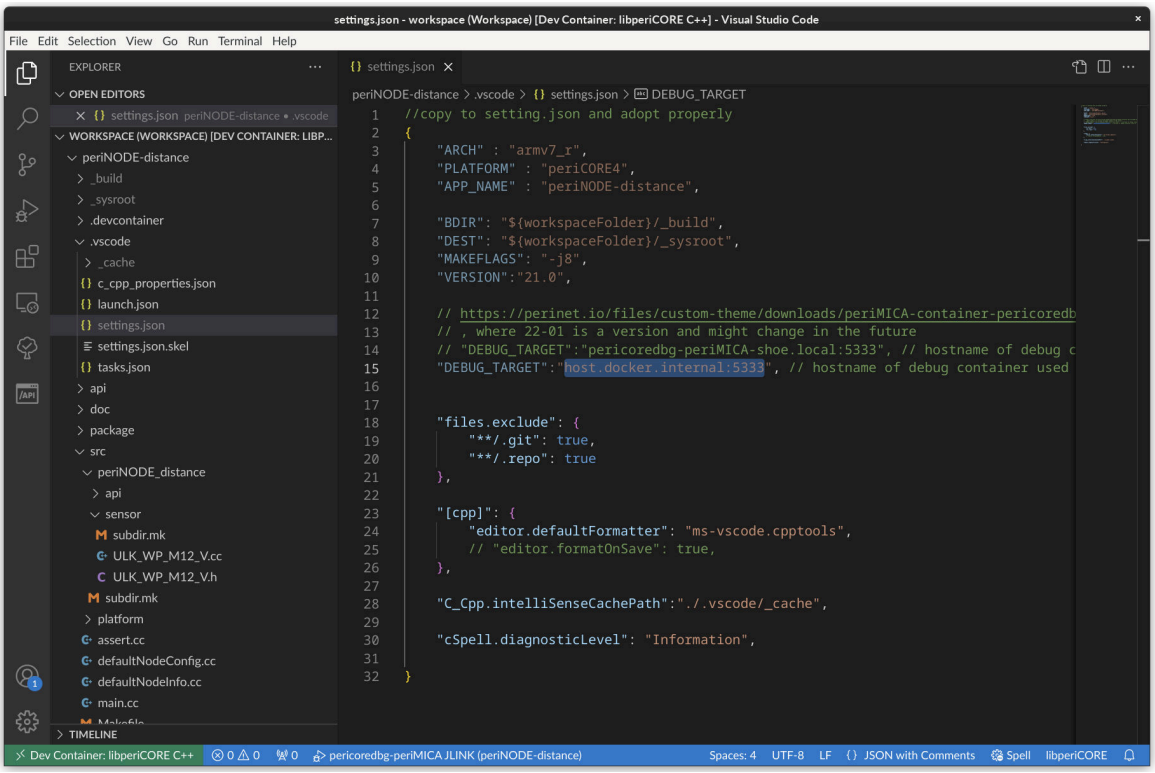


Figure 37: macOS configure settings.json

The special DNS (`host.docker.internal:5333`) resolves always to an IPv4 address, even though your docker is configured to work with IPv6. Because of this, we still need to forward the coming traffic on the specific TCP port4 5333 on the host machine to the *pericoredbg* container, which by default provides IPv6 only.

To forward the TCP4 port towards the *pericoredbg* container, we use the `socat` tool:

```
socat TCP4-LISTEN:5333,fork,reuseaddr \
  TCP6:pericoredbg-periMICA-serno.local:5333
```

Listing 13: Socat forwards IPv4 to IPv6 port

Note: Don't forget to replace the *pericoredbg-periMICA-serno* with your specific container name.

This command can be ran in a simple terminal, but this solution is not persistent as it needs to be always executed when working with the container. To reduce this effort, the command can be included in the login time, with the reminder that afterwards the port 5333 will be always forwarding the traffic to the container unless we stop running the command. To include the command in your login time, follow the steps below:

1. Start the application **Automator.app**
2. Select **Application**

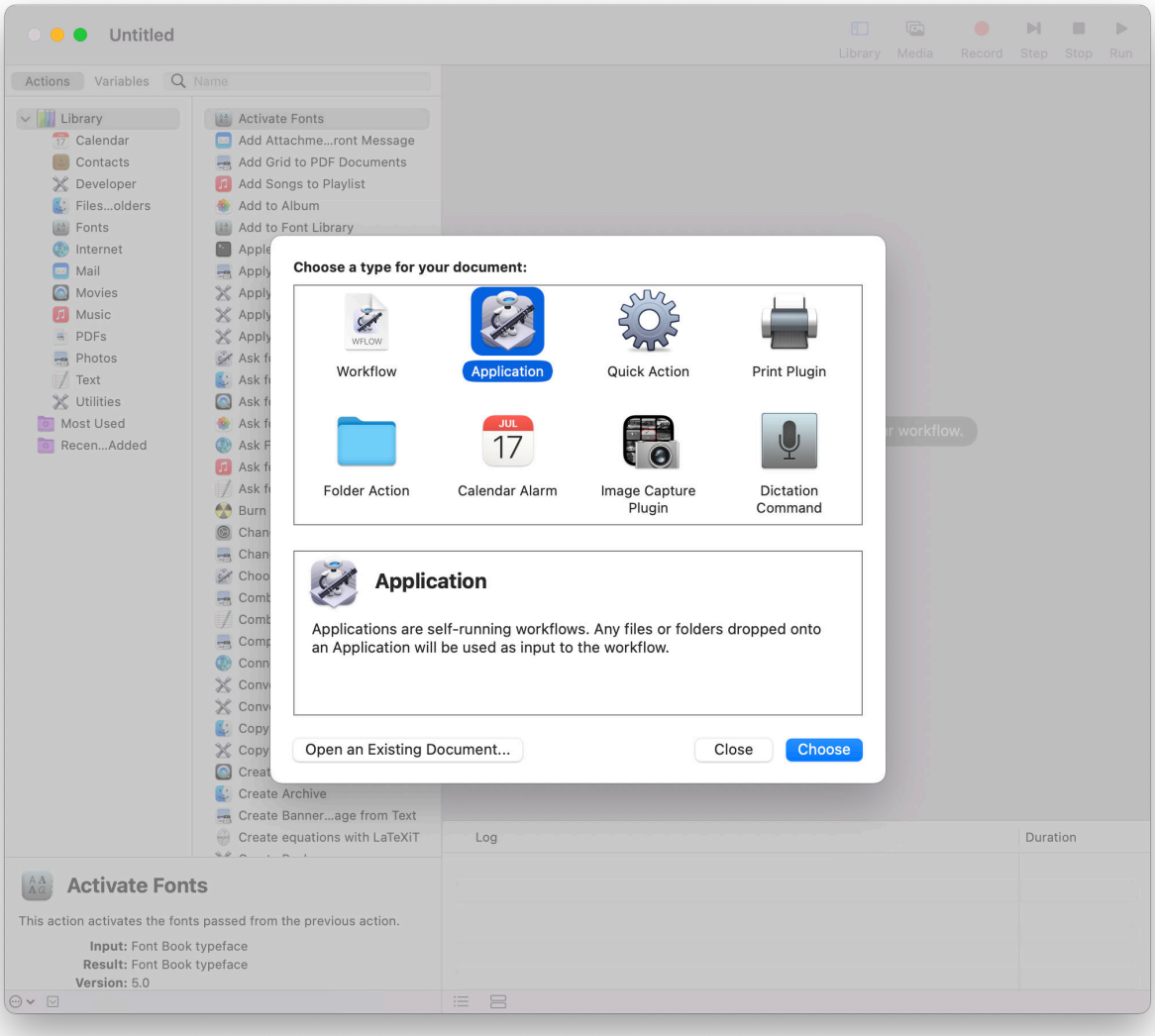


Figure 38: macOS Application

3. Under **Library**, select **Utilities** and **Run Shell Script**

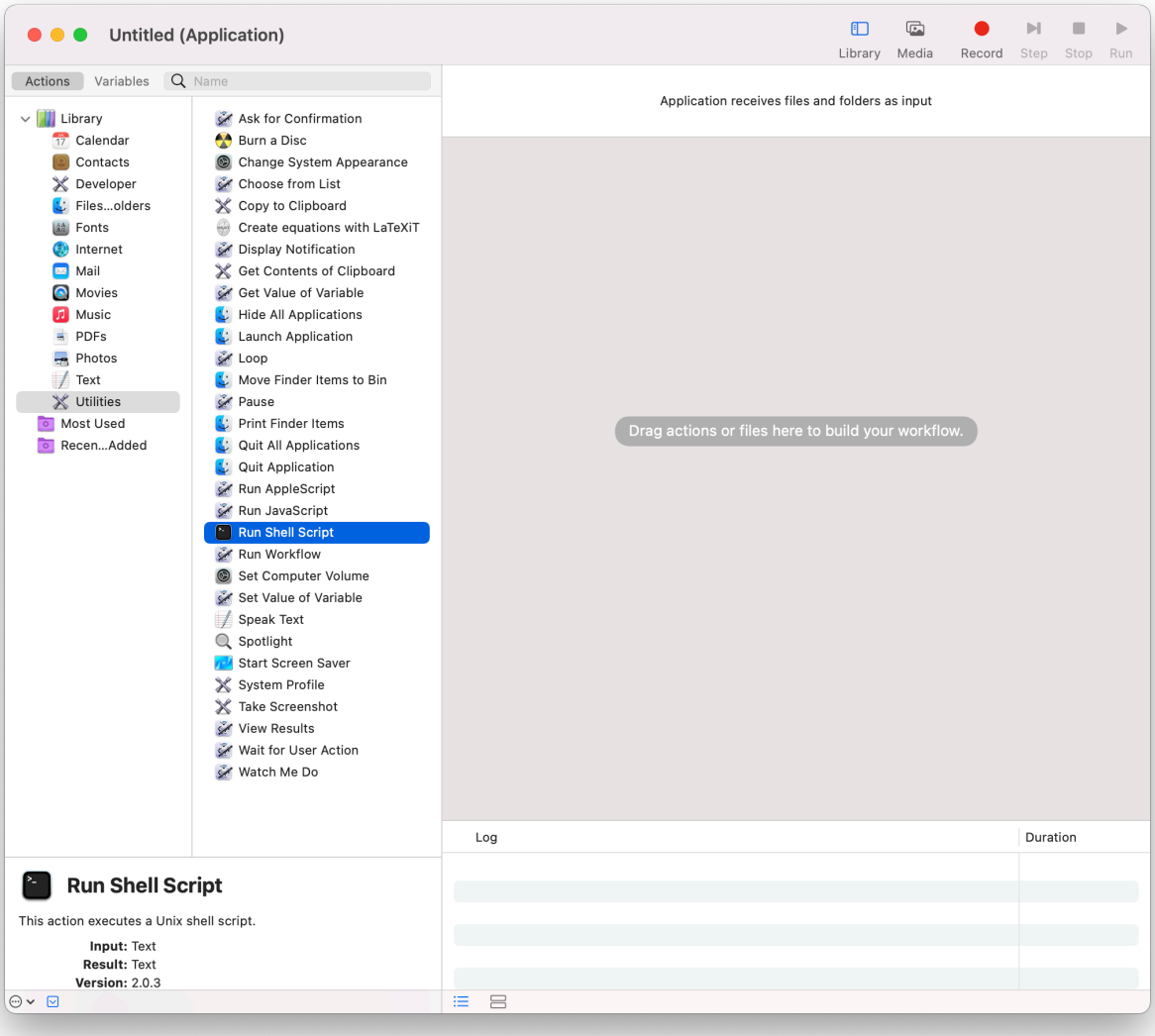


Figure 39: macOS Utilities

4. Copy the command and test it by clicking on the **play** button. Remember to change your *pericoredbg* container name.

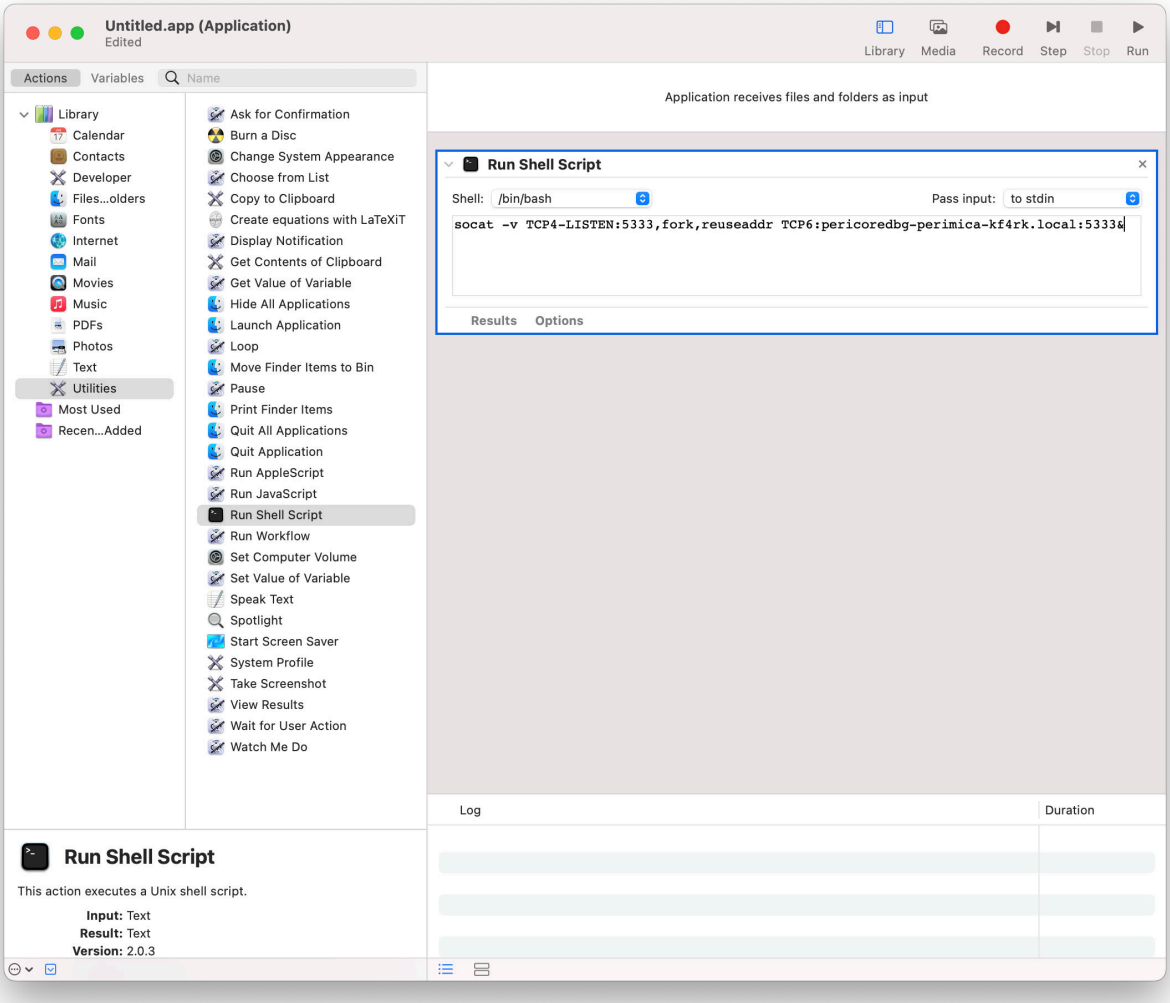


Figure 40: macOS Run Shell Script

5. If the command runs without problems, you can save the new application

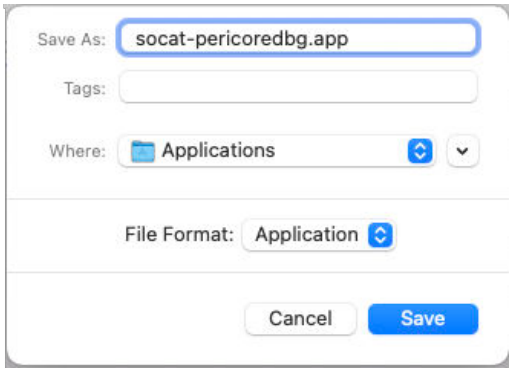


Figure 41: macOS Save Application

6. Now we can schedule the application to run when you log in. For this open **System Preferences...**

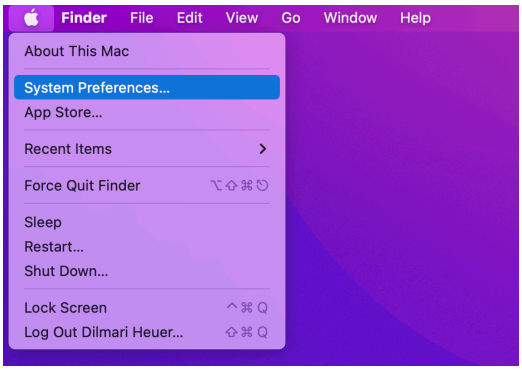


Figure 42: macOS System Preferences

7. Click on **Users & Groups** and include the application you saved in step 5.

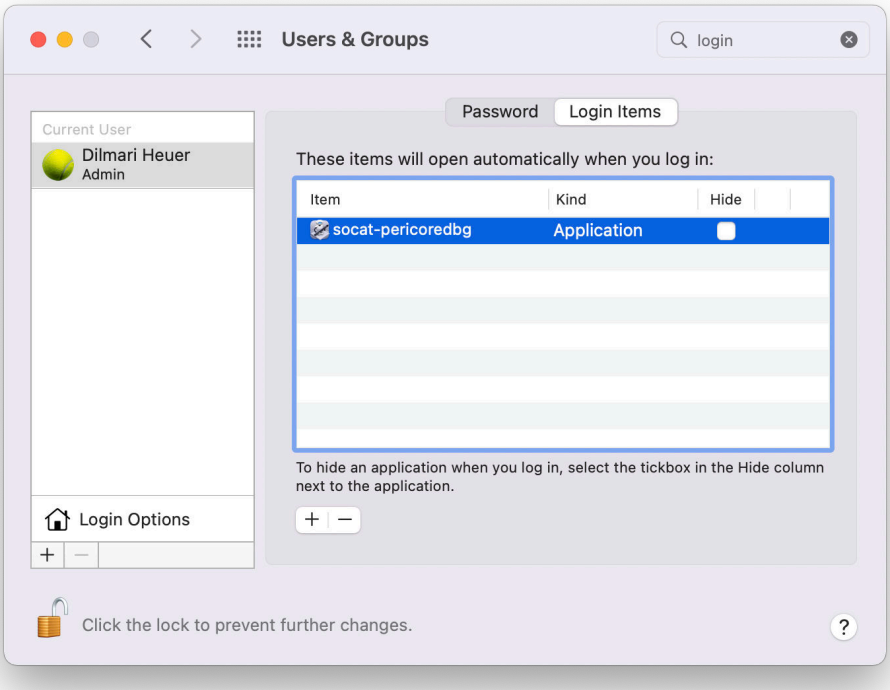


Figure 43: macOS Users & Groups

5.9 Specific Configuration for Windows

The docker network developed for Windows is implemented with a virtual machine and it does not create a **docker0** interface like in *Linux* based systems. This interface helps the docker container to communicate with the host network directly.

Another disadvantage of Windows Docker implementation is observed when the special DNS name resolves only to the IPv4 address in the host. For further information please refer to https://docs.docker.com/desktop/windows/networking/.

Due to these limitations, we need to create a way to communicate with the host machine network and afterwards with *pericoredbg* container.

First of all we need to use the special DNS name `host.docker.internal:5333` in order to communicate with the host machine. Configure it in your `settings.json` file through the variable `DEBUG_TARGET` as showed in Figure 44:

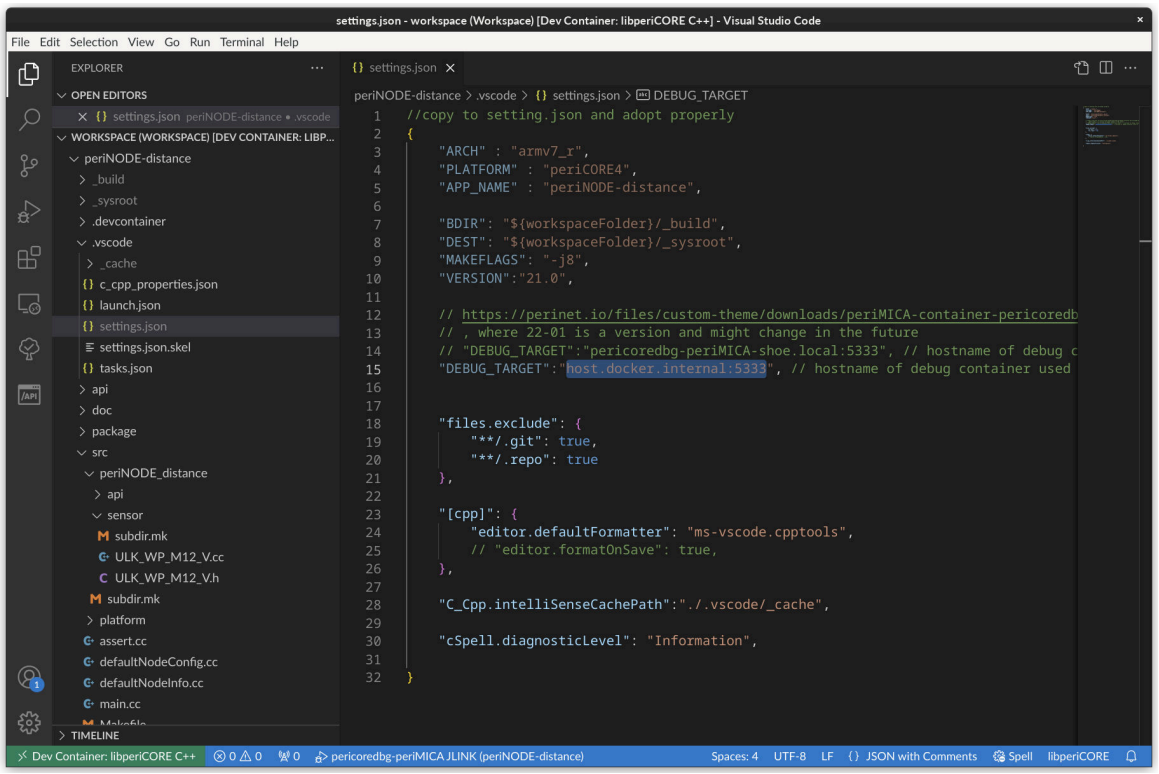


Figure 44: Windows configure settings.json

The special DNS (`host.docker.internal:5333`) resolves always to an IPv4 address even though your docker is configured to work with IPv6. Because of this, we still need to forward the coming traffic on the specific TCP port 5333 on the host machine to the *pericoredbg* container, which by default provides IPv6 only.

To forward the TCP4 port towards the *pericoredbg* container we use the `netsh` tool. Run a Powershell as Administrator:

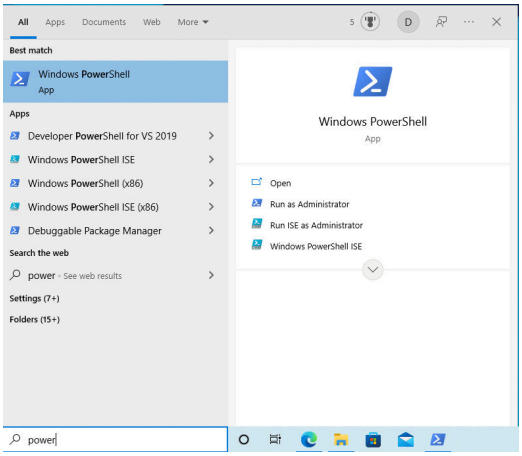


Figure 45: Windows open PowerShell

Run the following command (see Figure 46), but don't forget to replace the `pericoredbg-periMICA-serno` with your specific container name:

```
netsh.exe interface portproxy add v4tov6 listenport=5333 \
  connectaddress=pericoredbg-perimica-serno.local connectport=5333
```

Listing 14: Netsh forwards IPv4 to IPv6 port

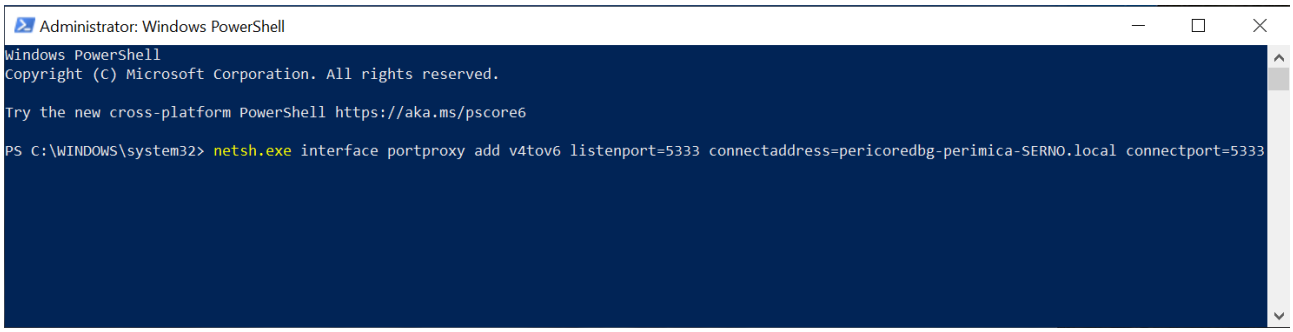


Figure 46: Windows forwarding port

In order to confirm the port forwarding you can check with following command:

```
netstat -ano | findstr :5333
```

Listing 15: Check the port forwarding IPv4 to IPv6 port

5.9.1 Open Docker Desktop Engine

An issue was observed when using IPv6 on Docker Desktop for Windows and trying to open it using the default variable ”experimentals”. If you have problems starting the engine, please use the configuration with the parameters shown in Figure 47:

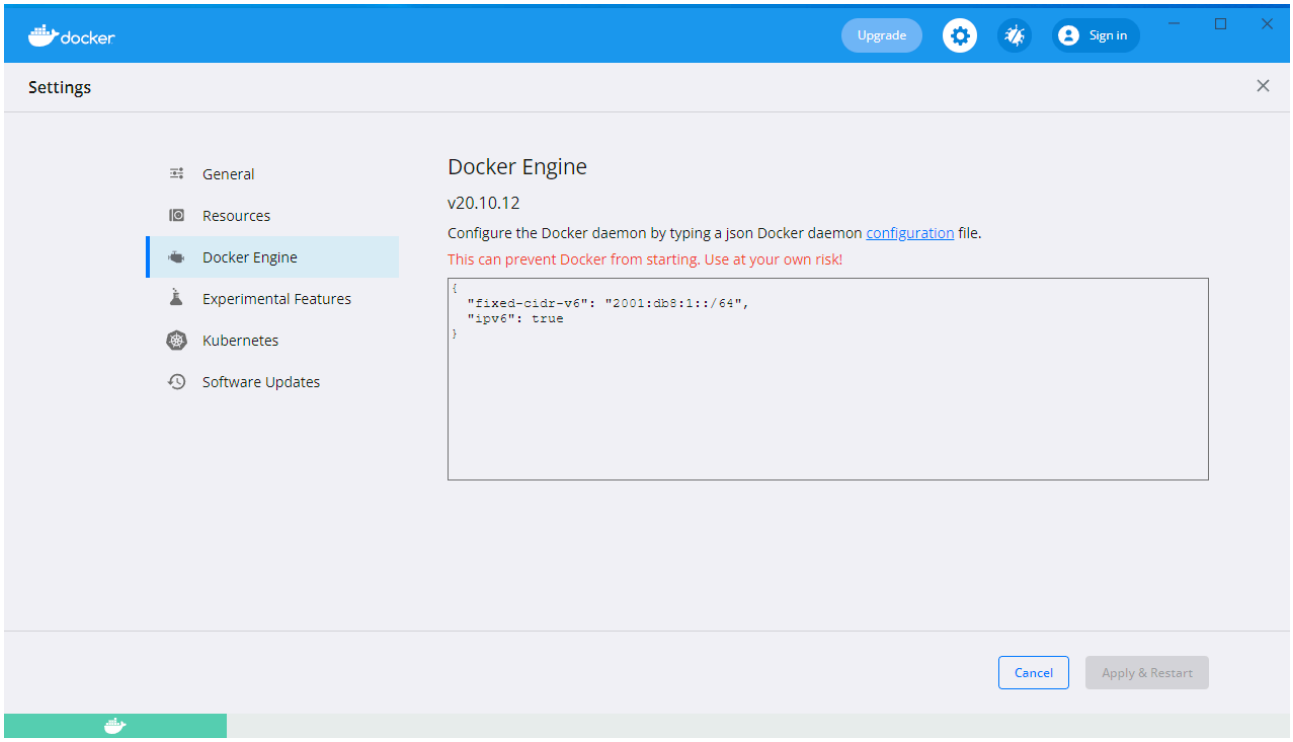


Figure 47: Windows Docker Config

5.9.2 Include code in PATH

Usually you are asked to include VS Code in `PATH` during installation. In case this was not done, but you want to start VS Code using the command line, this is possible at any time through the `PowerShell`, by simply running this command (Figure 48):

```
$Env:PATH += ";C:\Program Files\Microsoft VS Code\bin"
```

Listing 16: Include VS Code in PATH

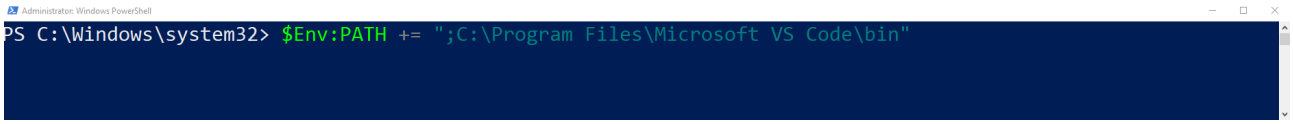


Figure 48: Windows include VS Code in PATH

5.10 Specific Configuration for Linux

Most Linux-based software distributions provide the Docker Desktop software through their package management system. Detailed installation instructions can be found at https://docs.docker.com/desktop/setup/install/linux or in the documentation of your preferred Linux distribution.

5.10.1 Configuring the Debug Server Target

By default, Perinet's development containers are set to use the same network interface as the host system. In Linux-based development environments, this configuration enables seamless access to the debug server running inside the `periMICA-container-pericoredbg`.

```
...

/// default: "DEBUG_TARGET": "host.docker.internal", // used with docker-desktop
/// override on linux
"DEBUG_TARGET":"pericoredbg-periMICA-sernm.local:5333", // hostname of the periCORE
  debug container, the remote debugserver

...
```

Listing 17: Settings file for accessing the debug container `.vscode/settings.json`

As illustrated in Listing 17, the variable `DEBUG_TARGET` specifies the hostname and port where the debug server for the periCORE Development Board is running. In this example, `pericoredbg` represents the container name, while `periMICA-sernm` refers to the hostname of the periMICA host system, which has a unique serial number (`sernm`).

5.10.2 Resolving IPv6 Addresses via mDNS

A Linux-based development environment typically relies on the glibc name resolution mechanism to resolve multicast DNS (MDNS) names, such as `pericoredbg-periMICA-sernm.local`. Perinet has observed that many distributions are configured incorrectly, limiting resolution to legacy IP addresses (IPv4). The following command (Listing 18) can be used to verify whether the system supports mDNS name resolution correctly.

```
user@linux-devenv:~ $ getent ahosts pericoredbg-perimica-sernm.local
fe80::762e:db02:3d8b:0%18 STREAM pericoredbg-perimica-sernm.local
fe80::762e:db02:3d8b:0%18 DGRAM
fe80::762e:db02:3d8b:0%18 RAW
```

Listing 18: Command to verify mDNS-based name resolution in a Linux distribution.

By default, name resolution is handled by the standard libc library, such as `glibc`. When using `glibc`, a dispatcher distinguishes mDNS queries from general DNS queries, directing them to the local network instead of a centralized DNS server.

mDNS queries are processed by user-space software. Popular implementations include `avahi` and `systemd`. These libraries are available through the package management systems of Linux distributions.

mDNS Name Resolution via Avahi

The `avahi` software package (https://avahi.org) integrates with the mDNS dispatcher in `glibc`, which detects mDNS name resolution requests and forwards them to the `avahi-daemon`. However, the default configuration in `/etc/nsswitch.conf` may prevent resolution of IPv6 addresses. The excerpt below (Listing 19) shows how to modify this configuration.

A misconfigured system may use `mdns4_minimal`, which limits resolution to legacy IP addresses (IPv4). To enable IPv6 name resolution, replace it with either `mdns_minimal` or `mdns6_minimal`.

```
#...
passwd:     files sss systemd
shadow:     files
group:      files [SUCCESS=merge] sss [SUCCESS=merge] systemd
#hosts:      files myhostname mdns4_minimal [NOTFOUND=return] dns
hosts:      files myhostname mdns_minimal [NOTFOUND=return] dns
services:   files sss
#...
```

Listing 19: Configuration file `/etc/nsswitch.conf` for avahi-based mDNS resolution.

mDNS Name Resolution via systemd

The `resolved` daemon, part of the systemd package, also supports mDNS name resolution. To enable this feature, (1) the `resolved` daemon must be properly configured, and (2) the glibc resolve dispatcher may require updating.

The following excerpt (Listing 20) demonstrates how to configure `resolved` to enable mDNS resolution.

```
[Resolve]
#DNS=
#FallbackDNS=
#Domains=
#DNSSEC=no
#DNSOverTLS=no
MulticastDNS=yes
#LLMNR=resolve
#Cache=yes
#CacheFromLocalhost=no
#DNSStubListener=yes
#DNSStubListenerExtra=
#ReadEtcHosts=yes
#ResolveUnicastSingleLabel=no
#StaleRetentionSec=0
```

Listing 20: Configuration file `/etc/systemd/resolved.conf` with enabled mDNS resolution.

To verify that name resolution via the `resolved` daemon is active, check the following configuration in `/etc/nsswitch.conf` (Listing 21).

```
#...
passwd:     files sss systemd
shadow:     files
group:      files [SUCCESS=merge] sss [SUCCESS=merge] systemd
#hosts:      files myhostname mdns4_minimal [NOTFOUND=return] dns
hosts:      files myhostname resolve [!UNAVAIL=return] mdns_minimal [NOTFOUND=return]
  dns
services:   files sss
#...
```

Listing 21: Configuration file `/etc/nsswitch.conf` for systemd-based mDNS resolution.

6 Application Deployment

In this section we intend to show the basic functionalities of the tools through building an example distance application. You will find here the basic process in order to build, debug and create images to flash the *periCORE SPE communication module*.

6.1 Locate the periCORE Hostname

The development board has the *periCORE SPE communication module* soldered and you need to know its hostname. The periCORE hostname is unique and it is based on the product serial number and its MAC Address and IPv6 link local address is also generated based on that. More detailed information about periCORE naming can be found in periCORE Datasheet [6].

The goal now is locate the periCORE hostname in the development board, and it can be easily find in the tag attached to the development board as shown in figure below:

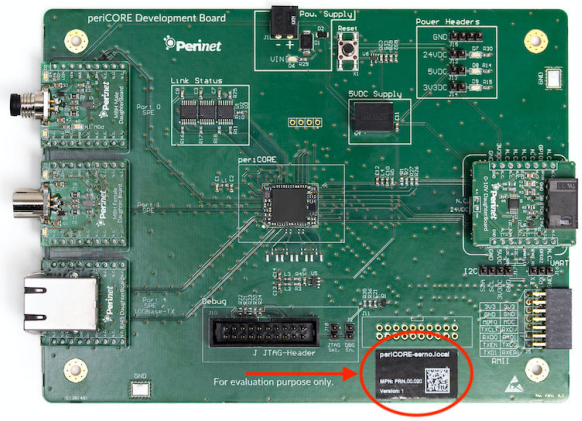


Figure 49: periCORE hostname tag

The examples of this guide refer to *periCORE-serno* as an example of hostname like you can see on the Figure 49. This also means that the services provided by periCORE would be accessible through the URL `https://periCORE-serno.local`.

6.2 Building the Example Distance Application

The building process of applications is automated through the `make` using `Makefiles`. In the *periNODE-distance* package there are two pre-configured tasks to help to build the applications, that can be accessed using the **Ctrl + Shift + B** shortcut (Windows and Linux), see Fig-

ure 50. Alternatively, you can issue the `make` command in the application folder and, it will build and link all the dependencies.

Note: macOS users should use **Command + Shift + B** .

The pre-configured tasks available in the workspace are:

firmware image: builds the firmware binary for debbuging purpose only. The final files will be located in `DEST` folder.

webui image: builds the webUI image which includes the optimized HTTP resources like .html, .css, etc. The final file will be located in `DEST` folder.

update-image: builds the firmware binary and creates an image file that can be flashed into the *periCORE SPE communication module*. The final files will be located in `DEST` folder.

package: similar to the update-image but the update image and the changelog files are compacted into a .zip package that can be used through the Update Web Interface to update the module. The final file will be located in `DEST` folder.

distclean: erase the content of the `BDIR` and `DEST` folders.

Note: Please verify the proper configuration of `DEST` folder.

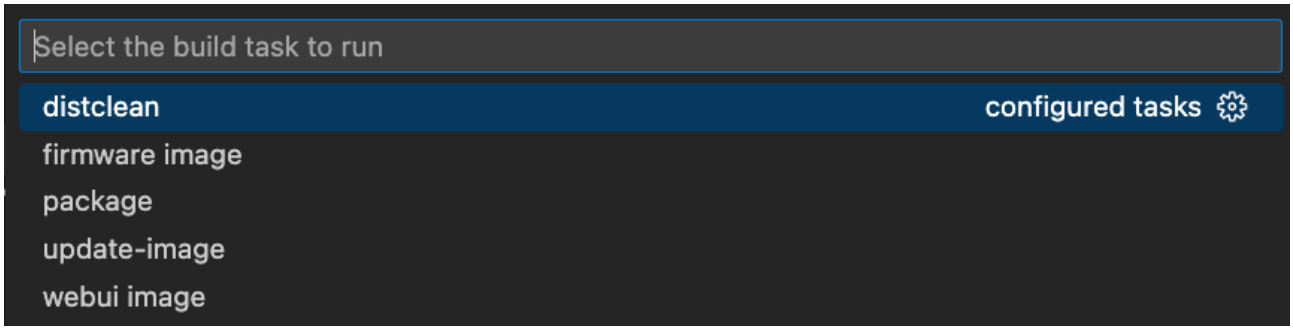


Figure 50: VS Code select build task

Please refer to Section 6.5 for details on how to update the *periCORE module* with the newly created image.

6.3 Debugging Using the periMICA

The execution of the application is controlled by the JTAG debugger, which allows stopping the program and following the code lines step-by-step, when the set breakpoint is hit. Registers and variables values can be checked. Simply select the options available on the *Run* menu.

Note: Only two hardware break points are supported.

- Click on the left side of a code line to setup a breakpoint (see Figure 51).
- Run the application with F5 key or through the Menu **Run →Start Debugging**.

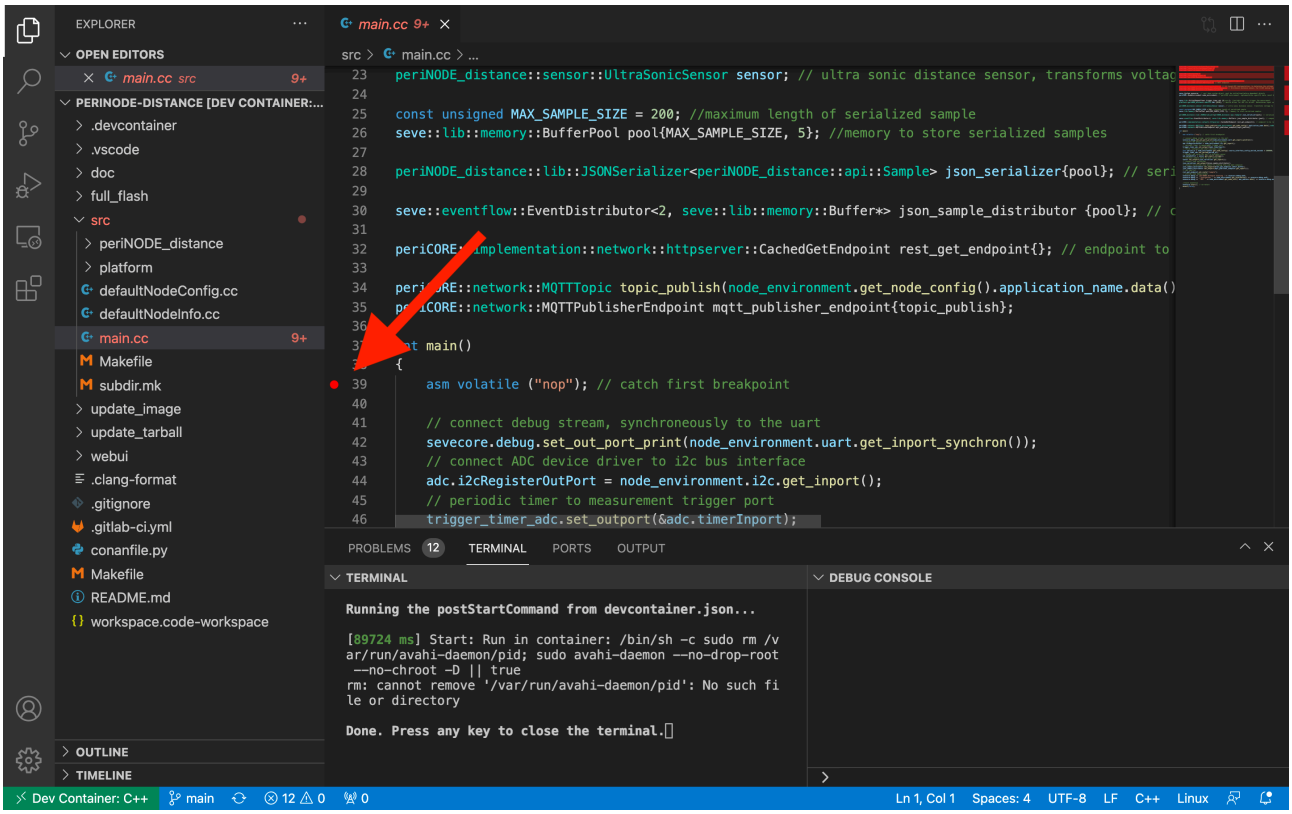


Figure 51: VS Code breakpoint

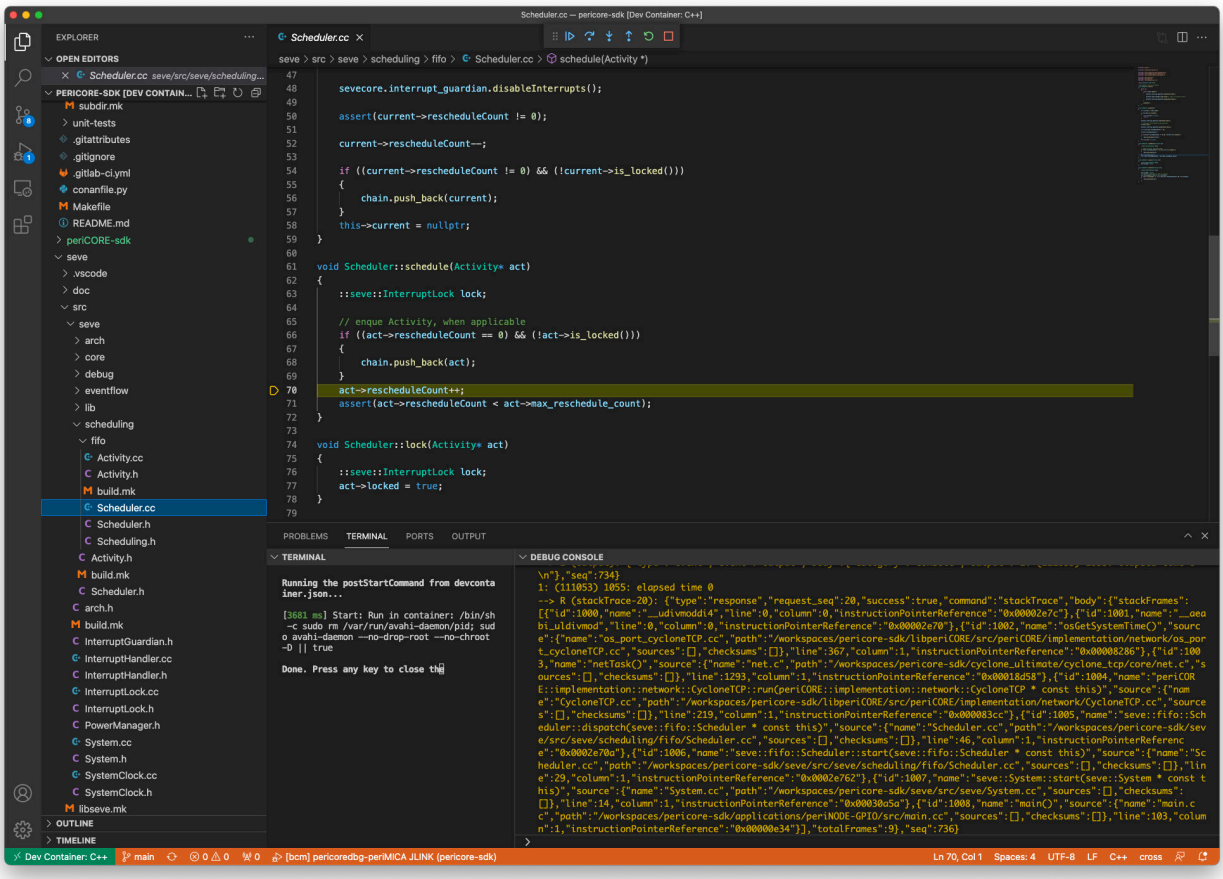


Figure 52: VS Code debugging

Note: A debug session via *JTAG* does download the application code into the volatile RAM memory. A power cycle or soft reboot will boot the persistent stored firmware image. For storing the application firmware in the persistent memory of the periCORE SPE communication module, please refer to section Section 6.5.

When trying to debug, make sure the *pericoredbg* container is running and if it is correctly configured in *settings.json* as explained in section 5.7.1. If there is a mistake in this configuration and the container is not achievable, a message like presented in figure 53 can be seen.

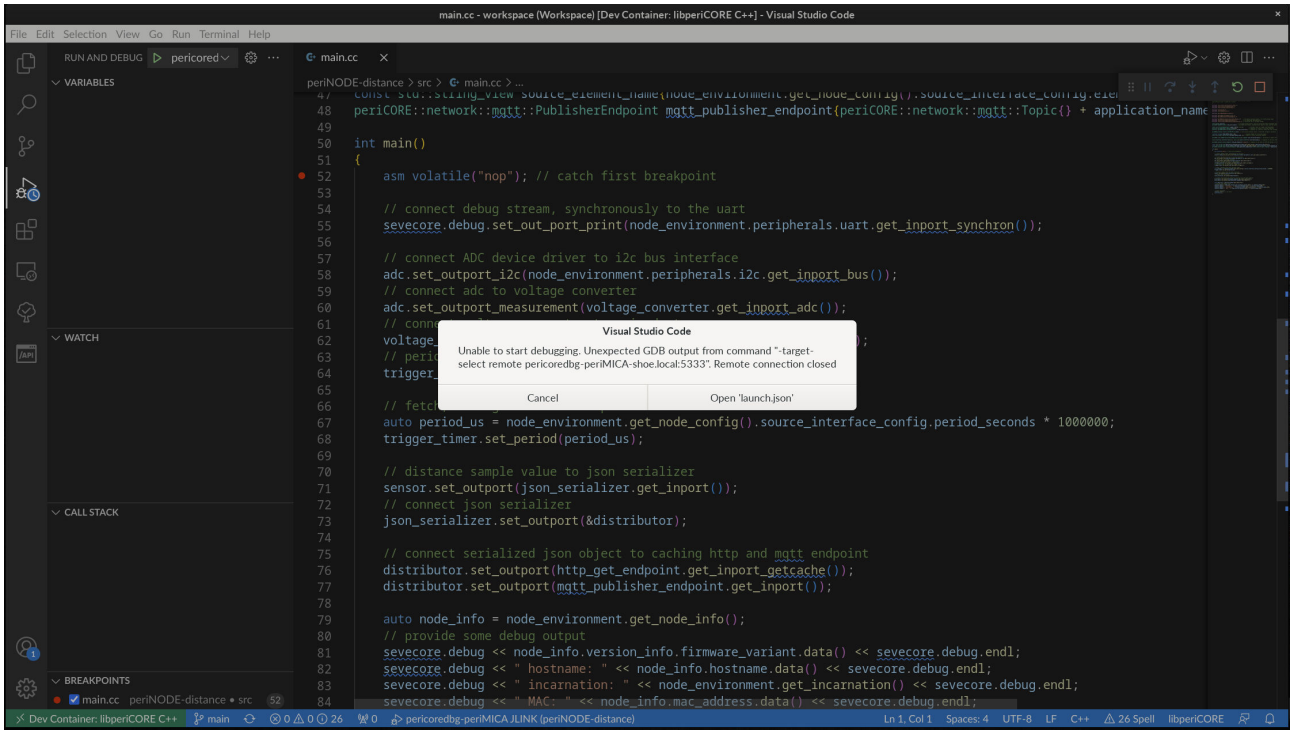


Figure 53: VS Code debugging error

If the container is connected and achievable but for any other reason it was not possible to debug, the message like shown in figure 54 will pop up.

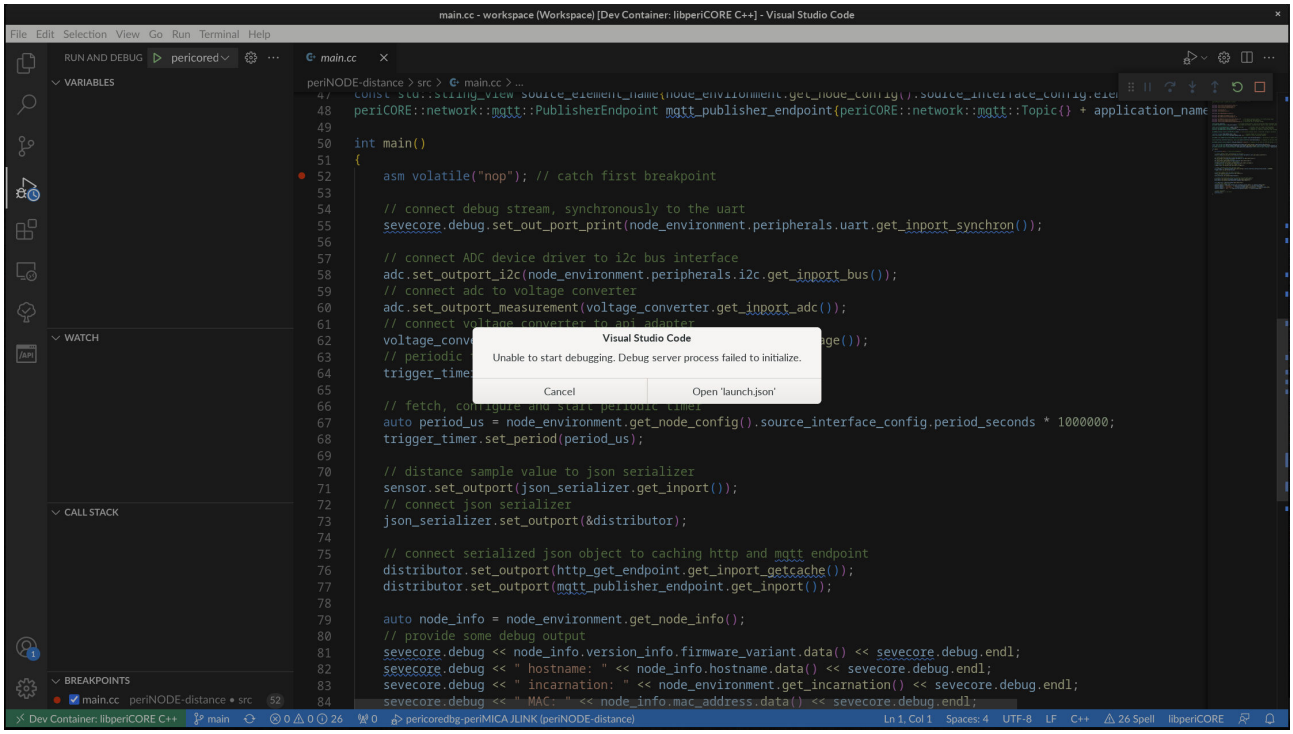


Figure 54: VS Code debugging

Note: Take advantage of the *pericoredbg* container log box to check if the connection was established and to see the errors in the GDB server.

6.4 Testing Basic Functionalities

Once periCORE is running, the device is announced in the network and the network connectivity can be tested with `ping`:

```
$ ping periCORE-serno.local
PING periCORE-serno.local(periCORE-serno.local (fe80::742e:db03:78aa:0)) 56 data bytes
64 bytes from periCORE-serno.local (fe80::742e:db03:7800:0%eth0): i1.0 seq=1 ttl=64
  time=0.258 ms
64 bytes from periCORE-serno.local (fe80::742e:db03:7800:0%eth0): i1.0 seq=2 ttl=64
  time=0.241 ms
```

Listing 22: Test periCORE using ping

```
$ avahi-browse -a
+   eth0 IPv6 periCORE-serno                                _https._tcp          local
```

Listing 23: DNS service discovery on Linux

On macOS, the MDNS DNS-SD is implemented in a native Bonjour tool. Simply run:

```
$ dns-sd -B _https._tcp
Browsing for _https._tcp
DATE: ---Wed 26 Jan 2022---
11:56:39.992  ...STARTING...
Timestamp     A/R    Flags  if Domain               Service Type         Instance Name
11:56:39.994  Add        2   6 local.               _https._tcp.          pericore-
  serno
```

Listing 24: DNS service discovery browse _https._tcp service instance

The http server implemented in periCORE is also started and the RestFul API provides the default URLs. The development board *periCORE module* come from factory with the periNODE-distance *image* flashed in the OEM partition. Because of this, it is possible to access the webui resources in the module through a Browser. Please refer to [6] for more information about the APIs and the life cycle of *periCORE module*.

Please find below an example of the `GET` method output for the url */info* using the `httpie` tool.

```
  $ http https://periCORE-serno.local/info --verify no
  HTTP/1.1 200 OK
Connection: close
Content-Type: application/json

{
    "hostname": "periCORE-serno",
    "mac_address": "76:2E:DB:03:78:00",
    "manufacturer": "Perinet GmbH",
    "pericore_charge": "123456",
    "pericore_part_number": "123456",
    "pericore_serial": "123456",
    "pericore_version": "3.3",
    "product_charge": "123456",
    "product_name": "periCORE Development Board",
    "product_part_number": "123456",
    "product_serial": "serno",
    "product_version": "",
    "version_info": {
        "firmware_variant": "periNODE distance",
        "firmware_version": {
            "api": 11,
            "build": 1642164041,
            "version_number": 18
        }
    }
}
```

Listing 25: GET Request on /info

In Figure 55 we can see the same output in the browser:

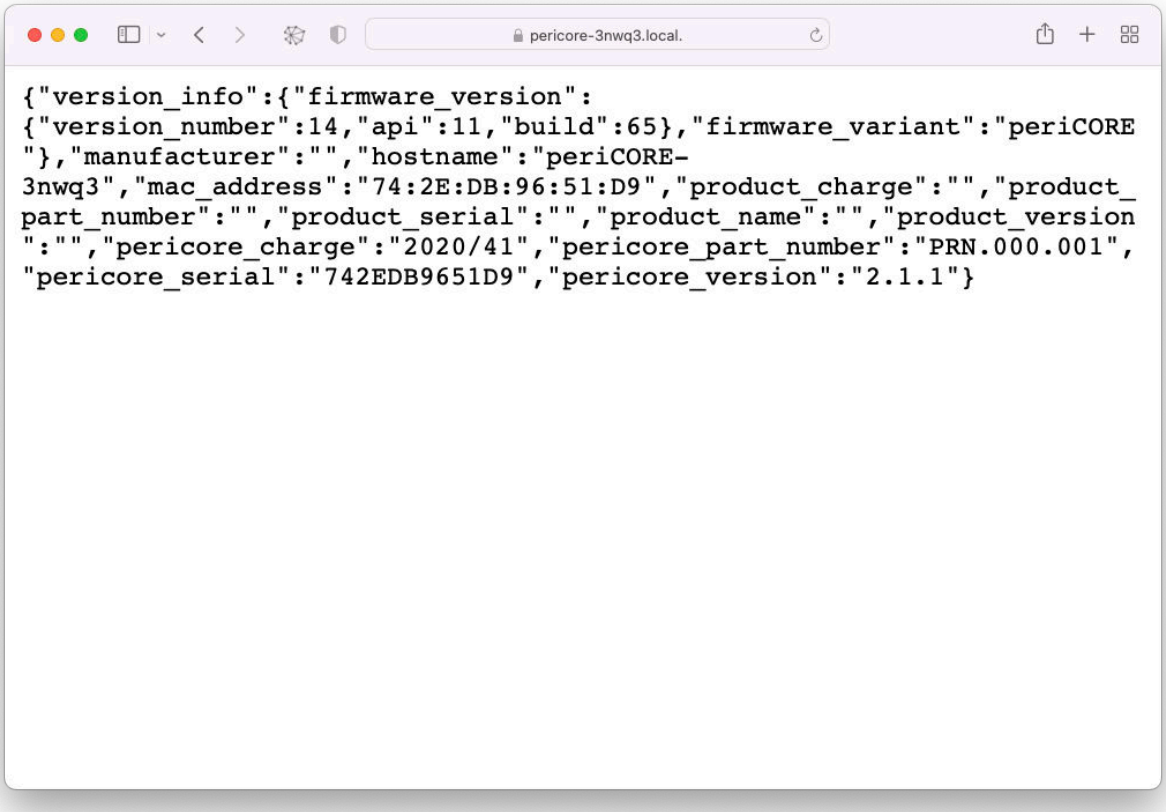


Figure 55: Accessing the API with the browser

6.5 Flashing Applications to periCORE module

Until now we developed and debugged the application with JTAG in the RAM memory. To store the application in the persistent flash memory, we need to build the `update` image file and copy it in the *periCORE* flash memory, by following these steps:

- Execute the task *app.perinode-distance update_image* as described in Section 6.2.
- Take the `update` image file from *_sysroot/bin* folder. The name is similar to `update_periNODE_distance-periCORE4-armv7_r-18.0.update.img.signed`.
- Transfer the update image to the *periCORE* flash memory with the help of `curl` and `PUT` method:

```
$ curl -6 -g --interface eth0 -X PUT
 "https://periCORE-serno.local/update"
 --data-binary '@update_periNODE_distance-periCORE4-armv7_r-18.0.update.img.signed'
 -H 'Expect:' --insecure --verbose
```

Listing 26: Initial Update periCORE using curl

After flashing the *update image* into the periCORE the bundle which contains the Web UI partition is also included and the *HTML* resources can be accessed through a browser.

6.6 Reset default periCORE configuration

The *periCORE* module on the development board is provided with an according *periCORE* application. The firmware brings its own default configuration which is stored in the flash memory of the *periCORE* module after bootup or when the user changes it.

Figure 56: Reset configuration using the *periNODE* Web UI.

In order to use the correct configuration for a specific *periNODE* or in general *Smart Component* application the default settings need to be reset once (Figure 56 or for command line see Listing 27).

```
$ curl -6 -g --interface eth0 -X PATCH "https://periCORE-serno.local/config/reset" --
  insecure
```

Listing 27: Config reset periNODE application using curl.

Immediately after reset the configuration parameters will not be taken from flash memory (which stored the *periCORE* configuration) but instead default values are used from the according application firmware. This will ensure that for *periNODE* devices sensor configuration is correctly setup, e.g. for publishing sensor data via MQTT.

6.7 Creating a New Front-end for the Application

The *webui* folder contains the necessary `Makefile` and http resources (`.html`, `.css`, `.js` files). The directory tree looks like in Figure 57:

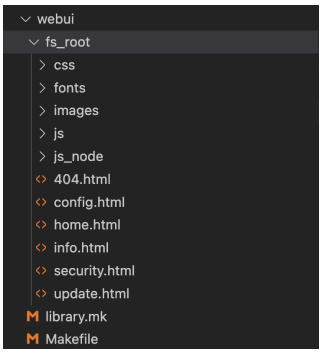


Figure 57: webui folder

Note: The *webui* storage partition is limited to 254 KB. When the limit is reached, you will get an error during the build process.

It is not possible to investigate the *webui* with the JTAG debugger. The behavior of the *webui* code can be observed in a web browser, after the *webui* image has been transferred to the webui partition of *periCORE*.

Please refer to periCORE Firmware Development Application Note [9] for more details.

At this point, you should have a basic overview on how to develop, debug and test your applications using the *periCORE development Kit*.

For further information on *Perinet IIoT* solutions please see the next chapter.

7 Further Readings

The goal of this application note was to provide guiding steps for having the complete setup of the *periCORE Development Kit*. More information about the *Perinet Smart Components* and the *periCORE SPE communication module* or 3rd-party tools described in this guide can be found online on https://www.perinet.io.

The most important documents that we recommend as further readings are listed below:

periCORE Datasheet [6]: Reference document of the periCORE module. It contains all the technical specifications for hardware, firmware, production handling, ordering and the regulatory approvals.

periCORE Development Kit User Guide [8]: Reference manual for the periCORE Development Kit, which presents the hardware interfaces and the periCORE SDK architecture.

periCORE Firmware Development Application Note [9]: Document that presents a C++ example application (periNODE distance) and the usage of *libperiCORE*.

sève Operating System Datasheet [11]: Technical reference document that explains the *sève* OS used in the *periCORE-SDK*. It presents the architectural principles used and the scheduling framework, the components and activities as well as the memory management.

periMICA User Guide [10]: *periMICA modular edge computer* reference manual that contains the hardware and software architectures and use-cases.

Smart Components Datasheet [12]: Reference document for the Perinet IIoT Smart Components designed to connect analogue and digital sensors and actuators to the Local Area Network through SPE technology.

Visual Studio Code Documentation : Online documentation on https://code.visualstudio.com/docs

Docker Documentation : Online documentation on https://docs.docker.com

8 Contact & Support

For customer support, please call us at **+49 30 863 206 701** or send an e-mail to *support@perinet.io*.

For complete contact information visit us at www.perinet.io

A List of Figures

B List of Listings

C List of Tables

D Glossary

DNS-SD DNS Service Discovery [1] is a way of using standard DNS programming interfaces, servers and packet formats to browse the network for services. 66

HTTP Hypertext Transfer Protocol is an application-layer protocol for transmitting hypermedia documents, such as HTML. 61

IIoT Industrial Internet of Things. 70, 71

IPv4 Internet Protocol Version 4, a communication protocol. 48, 54

IPv6 Internet Protocol Version 6 [13], a communication protocol. 35, 48, 54, 60

JTAG Joint Test Action Group is a standard that defines tools to debug embedded systems. 29, 31, 62, 68, 70, 73

MAC Media Access Controller, component that handles concurrent access to a shared physical communication medium. 60

mDNS multicast Domain Name Service [2], a protocol that implements a local distributed name resolving mechanism. 57, 66

MQTT Message Queuing Telemetry Transport is a lightweight, publish-subscribe based network protocol that transports messages between devices. 70

OCI Open Container Initiative The Open Container Initiative is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.. 9, 22, 33

OS Operating System, a system software that manages computer hardware and software resources. 71

PoE Power over Ethernet is a network standard that defines the feature of ethernet cables to suplly power and data connection. 30

SPE Single Pair Ethernet. 60, 71

TCP Transmission Control Protocol. 48, 54

UI User Interface. 41, 69

URL Uniform Resource Locator. 60

USB Universal Serial Bus is a standard of the connection of peripherals to personal computers. 31

E References

[1] S. Cheshire and M. Krochmal. *DNS-Based Service Discovery*. RFC 6763. http://www.rfc-editor.org/rfc/rfc6763.txt. RFC Editor, Feb. 2013. URL: http://www.rfc-editor.org/rfc/rfc6763.txt.

[2] S. Cheshire and M. Krochmal. *Multicast DNS*. RFC 6762. http://www.rfc-editor.org/rfc/rfc6762.txt. RFC Editor, Feb. 2013. URL: http://www.rfc-editor.org/rfc/rfc6762.txt.

[3] Software Freedom Conservancy. *Git distributed version control system*. URL: https://git-scm.com.

[4] Perinet GmbH. OCI container for cross application development of periCORE based firmware. URL: https://gitlab.com/perinet/periCORE/applications/devcontainer.

[5] Perinet GmbH. periCORE based firmware example for the periNODE-0-10V. URL: https://gitlab.com/perinet/periNODE-0-10V/periNODE-distance-firmware.git.

[6] Perinet GmbH. periCORE Datasheet. PRN.100.375. https://docs.perinet.io/PRN100375-periCOREDatasheet.pdf.

[7] Perinet GmbH. periCORE debug server container for the periMICA. URL: https://gitlab.com/perinet/periMICA-container/pericoredbg.

[8] Perinet GmbH. periCORE Development Kit User Guide. PRN.100.378. https://docs.perinet.io/PRN100378-periCOREDevelopmentKitUserGuide.pdf.

[9] Perinet GmbH. periCORE Firmware Development Application Note. PRN.100.379. https://docs.perinet.io/.

[10] Perinet GmbH. periMICA User Guide. PRN.100.392. https://docs.perinet.io/PRN100392-periMICAUserGuide.pdf.

[11] Perinet GmbH. sève Operating System Datasheet. PRN.100.377. https://docs.perinet.io/.

[12] Perinet GmbH. Smart Components Datasheet. PRN.100.387. https://docs.perinet.io/PRN100387-SmartComponentsDatasheet.pdf.

[13] R. Hinden and S. Deering. *IP Version 6 Addressing Architecture*. RFC 4291. http://www.rfc-editor.org/rfc/rfc4291.txt. RFC Editor, Feb. 2006. URL: http://www.rfc-editor.org/rfc/rfc4291.txt.

[14] Docker Inc. *Docker*. URL: https://www.docker.com.

[15] Microsoft. *Visual Studio Code*. URL: https://code.visualstudio.com.

[16] Microsoft. *Visual Studio Code - Developing inside a Container*. URL: https://code.visualstudio.com/docs/remote/containers.

[17] Microsoft. *Visual Studio Code - Workspace Trust*. URL: https://code.visualstudio.com/docs/editor/workspace-trust.

F Revision History

Revision	Date	Author(s)	Description
1	2022-05-31	Engineering	Initial Release
2	2023-04-24	Engineering	Update URLs to gitlab.com/perinet; add quick start description to Section 3
3	2023-04-28	Engineering	add missing port for DEBUG_TARGET for macOS (Section 5.8) and Windows (Section 5.9).
4	2025-02-26	Engineering	Fix extra characters for listings to allow copy and paste, add section Section 5.10