

A212 SDK

VS Code User Guide

A212 SDK

VS Code User Guide

Author: Acconeer AB

Version: a2-v1.2.0

Acconeer AB Nov 03, 2026

© 2026 by Acconeer AB - All rights reserved

Contents

1 Introduction 4
1.1 Overview of the Files 4
1.2 Requirements 4
2 Install on Linux 5
2.1 Install Docker Engine 5
2.2 Install VS Code 5
3 Install on Windows 6
3.1 Install WSL 2 6
3.2 Install Docker Engine in WSL 6
3.3 Install VS Code 6
4 Open the SDK in the Dev Container 8
5 Build the Applications 10
5.1 Check the Toolchain 10
5.2 Build Everything 10
5.3 Build a Single Application 11
5.4 Clean 11
5.5 Code Navigation (IntelliSense) 11
5.6 Add Custom Code 12
6 Use the XE212 EVK (XE212 SDK Only) 13
6.1 Choose the Dev Container Configuration 13
6.2 Give the User Access to the XE212 EVK on Linux 13
6.3 Load and Flash on Linux 14
6.4 Debug with GDB on Linux 16
6.4.1 Debug in VS Code 16
6.4.2 Debug in the Terminal 17
6.5 Load and Flash on Windows 18
6.6 Troubleshooting 19
7 Tips and Tricks 20
7.1 Rebuild the Container 20
7.2 Leave the Container or Switch Configuration 20
8 Troubleshooting 21
9 Disclaimer 22

© 2026 by Acconeer AB - All rights reserved

Acconeer SDK Documentation Overview

To better understand which SDK document to use, a summary of the documents is shown in the table below. More general information about Pulsed Coherent Radar, PCR, can be found at docs.acconeer.com.

Name	Description	When to use
RSS API documentation (html)		
rss_api	The complete C API documentation.	• RSS application implementation • Understanding RSS API functions
User guides (PDF)		
A212 Sensor FW	Describes how to develop SW on the A212	• Developing SW inside the A212 sensor
A212 SDK VS Code	Describes how to build the A212 SDK and XE212 SDK, and use the XE212 EVK, in a VS Code dev container	• Working with A212 SDK or XE212 SDK in VS Code
XE212 SDK	Describes how to develop SW on the A212 using the XE212 EVK	• Working with XE212 SDK
Reference Manuals (PDF)		
A212 Technical	Complete information on A212 memory and peripherals	• Advanced development of SW inside the A212 sensor • Development of driver SW inside the A212 sensor
Application Notes (PDF)		
DFU Protocol	Defines the protocol used to communicate with the A212, either with UART or SPI in DFU mode	• Development of bootstrap software for external host
NVM Image Format	Defines the format of the image data to stored in external NVM	• Development of A212 software that loads from external NVM
Exploration Server Protocol	Defines the request/response protocol between clients and the Exploration Server	• Development of client software for the exploration server
Readme (txt)		
README	Various target specific information and links	• After SDK download
BUILDINFO	Compiler information	• To understand pre-compiled code

Table 1: SDK document overview.

© 2026 by Acconeer AB - All rights reserved

1 Introduction

This guide describes how to build the A212 SDK and the XE212 SDK with Visual Studio Code (VS Code) and a *dev container*. Everything except chapter 6 “Use the XE212 EVK (XE212 SDK Only)” applies to both SDKs.

A dev container is a Docker container that VS Code starts and connects to. It contains everything that is needed to compile and link the A212 applications:

- GNU `make`
- Version 5.4.1 of the AndeSight `nds32le-elf-mculib-v5f` toolchain (`riscv32-elf-gcc`)
- `uv`, used to run the Python scripts in the `script/` folder of the SDK
- `bear`, used to generate `compile_commands.json` for code navigation

Nothing except Docker and VS Code has to be installed on the computer. The SDK folder on the computer is shared with the container, so files edited in VS Code and files that are built (in the `out/` folder) are always the same on both sides.

1.1 Overview of the Files

The following files in the SDK root folder are used by VS Code. Folders that start with a `.` are hidden by default in most file managers.

File	Purpose
`.devcontainer/Dockerfile`	Describes the container image (Ubuntu 24.04, toolchain, `make`, `gdb-multiarch`, `uv`)
`.devcontainer/devcontainer.json`	The dev container configuration. Works on Linux and Windows
`.vscode/tasks.json`	Build and clean tasks (**Terminal > Run Task...**)
`.vscode/c_cpp_properties.json`	C/C++ IntelliSense settings for the A212 toolchain
`.vscode/settings.json`	Editor settings for the SDK (line endings, file associations)
`.vscode/extensions.json`	Recommended VS Code extensions

Table 2: Dev container and VS Code files

1.2 Requirements

- An **x86-64 (Intel or AMD)** computer. The AndeSight toolchain is only available for x86-64 Linux, so ARM based computers (for example Apple Silicon or Windows on ARM) are not supported.
- Linux, or Windows 10 (version 22H2 or later) or Windows 11.
- About 3 GB of free disk space for the container image.
- An internet connection the first time the container is created. The toolchain (about 250 MB) is downloaded from GitHub and the Python packages for the scripts in the SDK are downloaded from PyPI.

© 2026 by Acconeer AB - All rights reserved

2 Install on Linux

Ubuntu 22.04 and 24.04 are used as reference in this guide. Other Linux distributions work in the same way but the commands to install packages may differ.

2.1 Install Docker Engine

1. Open a terminal.
2. Install Docker Engine with the official convenience script. For other distributions, follow `https://docs.docker.com/engine/install/`. Do not install Docker as a Snap.

   ```
   $> curl -fsSL https://get.docker.com -o get-docker.sh
   $> sudo sh get-docker.sh
   ```

3. Add the user to the `docker` group, so that Docker can be used without `sudo`:

   ```
   $> sudo usermod -aG docker $USER
   ```

4. **Log out and log in again** (or restart the computer) so that the new group membership is applied.
5. Verify that Docker and its buildx plugin work:

   ```
   $> docker run --rm hello-world
   $> docker buildx version
   ```

   The output should contain `Hello from Docker!` followed by the buildx version.

2.2 Install VS Code

1. Download the `.deb` package from `https://code.visualstudio.com/download` and install it. Do not use the Snap or Flatpak version of VS Code.

   ```
   $> sudo apt install ./code_*_amd64.deb
   ```

2. Start VS Code.
3. Open the Extensions view (`Ctrl+Shift+X`), search for **Dev Containers** (published by Microsoft, id `ms-vscode-remote.remote-containers`) and click **Install**.

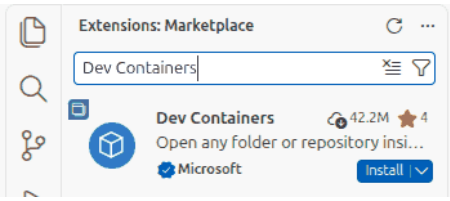


Figure 1: The Dev Containers extension in the Extensions view

Continue with chapter 4 “Open the SDK in the Dev Container”.

© 2026 by Acconeer AB - All rights reserved

3 Install on Windows

On Windows, Docker runs in *WSL 2* (Windows Subsystem for Linux), which is a Linux system inside Windows. The SDK is also stored in WSL, because building is much faster there than on a Windows drive.

3.1 Install WSL 2

1. Make sure that virtualization is enabled. Open **Task Manager**, select the **Performance** tab and click **CPU**. **Virtualization** should say *Enabled*. If it does not, enable it in the BIOS/UEFI settings of the computer (often called *Intel VT-x*, *Intel Virtualization Technology* or *AMD-V/SVM*).
2. Open **PowerShell as Administrator** (right-click the Start button and select **Terminal (Admin)** or **Windows PowerShell (Admin)**).
3. Install WSL with the default Ubuntu distribution:

   ```
   PS> wsl --install
   ```

4. Restart the computer when asked.
5. After the restart, an Ubuntu window opens and asks for a new UNIX user name and password. Choose any name and password. This user is only used inside WSL. If the window does not open, start **Ubuntu** from the Start menu.
6. Verify in PowerShell that Ubuntu runs on WSL version 2:

   ```
   PS> wsl --list --verbose
   ```

   The `VERSION` column should show `2`. If it shows `1`, run `wsl --set-version Ubuntu 2`.
7. The AndeSight toolchain crashes on the default WSL 2 kernel. Open Notepad and paste the following text:

   ```
   [wsl2]
   kernelCommandLine = vsyscall=emulate
   ```

   Save it as `.wslconfig` in the Windows user folder (`C:\Users\<windows user>`). Select **All files** as file type, so that Notepad does not add `.txt` to the name.
8. Restart WSL so that the setting is applied:

   ```
   PS> wsl --shutdown
   ```

3.2 Install Docker Engine in WSL

1. Start **Ubuntu** from the Start menu. A Linux terminal opens.
2. Follow section 2.1 “Install Docker Engine” in that terminal. Instead of logging out, run `wsl --shutdown` in PowerShell and then start **Ubuntu** again.

> **NOTE:** If Docker Desktop is installed, quit it and turn off **Settings > Resources > WSL integration** for Ubuntu, so that it does not replace Docker Engine in WSL.

3.3 Install VS Code

1. Download and install VS Code for Windows from `https://code.visualstudio.com/download`. In the installer, keep **Add to PATH** checked.
2. Start VS Code.
3. Open the Extensions view (`Ctrl+Shift+X`) and install these two extensions, both published by Microsoft:
   - **Dev Containers** (`ms-vscode-remote.remote-containers`)
   - **WSL** (`ms-vscode-remote.remote-wsl`)

© 2026 by Acconeer AB - All rights reserved

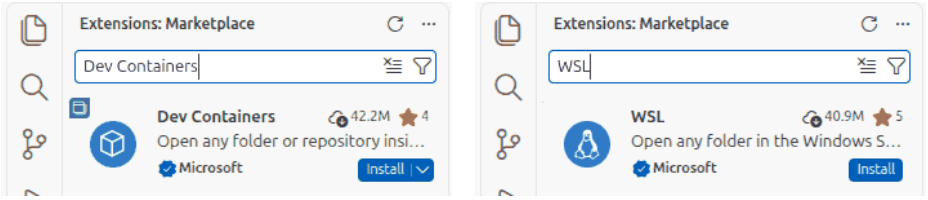


Figure 2: The Dev Containers and WSL extensions in the Extensions view

4. Open **File > Preferences > Settings**, search for `dev.containers.executeInWSL` and check it.

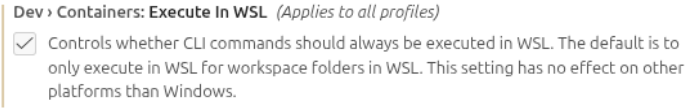


Figure 3: The `dev.containers.executeInWSL` setting

Continue with chapter 4 “Open the SDK in the Dev Container”.

© 2026 by Acconeer AB - All rights reserved

4 Open the SDK in the Dev Container

1. Open a Linux terminal. On Windows, start **Ubuntu** from the Start menu.
2. Unpack the SDK zip file to a folder of choice, for example `~/acconeer`. On Windows, the `Downloads` folder is found at `/mnt/c/Users/<windows user>/Downloads`.

   ```
   $> sudo apt update
   $> sudo apt install unzip
   $> mkdir -p ~/acconeer
   $> cd ~/acconeer
   $> unzip ~/Downloads/<sdk zip file>.zip
   ```

3. Find the SDK root folder. It is the folder that contains the `makefile` and the `source`, `script` and `.devcontainer` folders. In the rest of this guide that folder is called `<sdk folder>`.
4. Open the folder in VS Code from the same terminal:

   ```
   $> cd ~/acconeer/<sdk folder>
   $> code .
   ```

   On Windows, VS Code installs a small server in WSL the first time. VS Code opens on Windows and the lower left corner shows **WSL: Ubuntu**. From Windows, the folder is found in File Explorer under **Linux > Ubuntu > home > (WSL user) > acconeer > (SDK folder)**.
5. If VS Code asks *"Do you trust the authors of the files in this folder?"*, click **Yes, I trust the authors**.

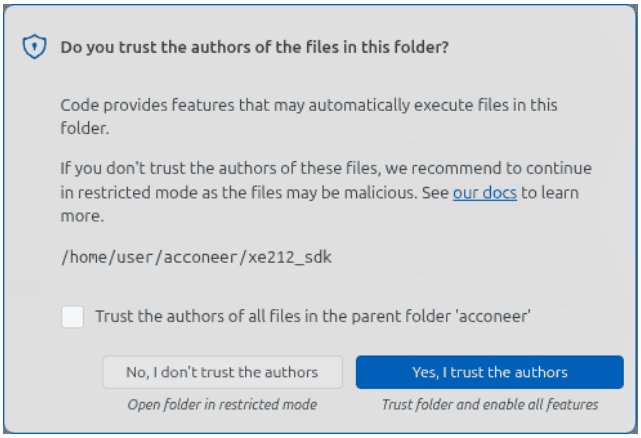


Figure 4: The workspace trust dialog

6. A notification *"Folder contains a Dev Container configuration file. Reopen folder to develop in a container"* is shown in the lower right corner. Click **Reopen in Container**.

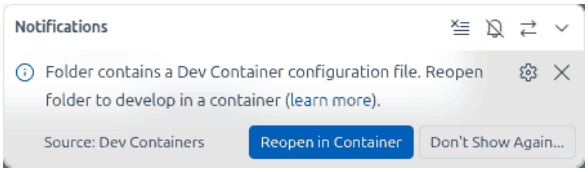


Figure 5: The Reopen in Container notification

   If the notification does not appear, press `F1` (or `Ctrl+Shift+P`), type `Dev Containers: Reopen in Container` and press `Enter`.

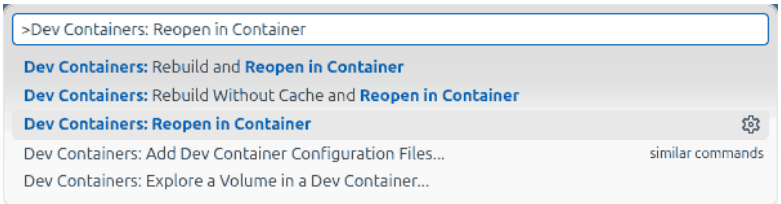


Figure 6: Reopen in Container from the Command Palette

7. The XE212 SDK has two dev container configurations, so VS Code asks which one to use. Select it as described in section 6.1 "Choose the Dev Container Configuration". VS Code remembers the choice. To change it later, see

© 2026 by Acconeer AB - All rights reserved

section 7.2 “Leave the Container or Switch Configuration”. The A212 SDK has one configuration, so VS Code does not ask.

8. Wait while the container image is built. This takes a few minutes the first time, because the toolchain is downloaded. The status bar shows **Configuring Dev Container (show log)** while the container starts. Click it to follow the progress. The next time the folder is opened, the container starts in a few seconds.

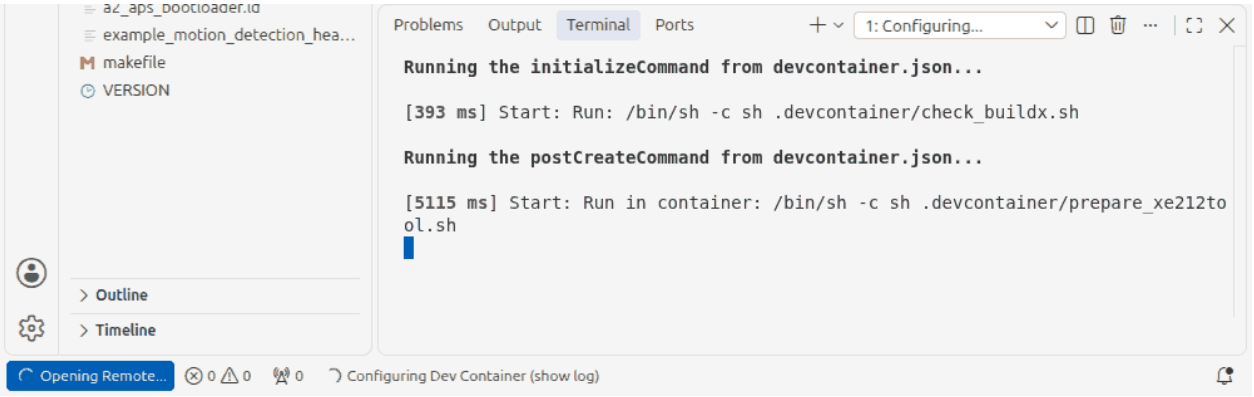


Figure 7: The dev container log while the container starts

9. When the container is ready, the lower left corner of VS Code shows **Dev Container:** followed by the name of the configuration.



Figure 8: The status bar in the A212 SDK and in the XE212 SDK (Linux host, USB access) containers

© 2026 by Acconeer AB - All rights reserved

5 Build the Applications

All commands in this chapter run **inside the dev container**, either from a VS Code terminal (**Terminal > New Terminal**) or as VS Code tasks (**Terminal > Run Task...**).

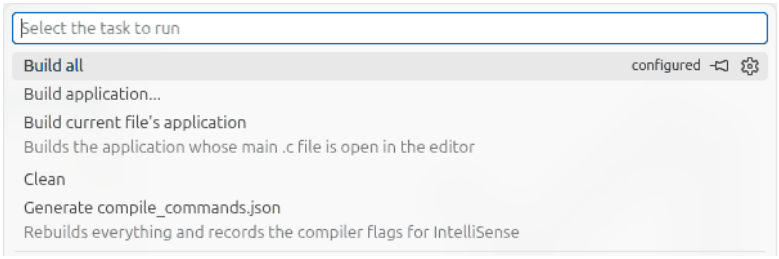


Figure 9: The tasks in **Terminal > Run Task...**

5.1 Check the Toolchain

Open a terminal with **Terminal > New Terminal**. The terminal is opened inside the container, which is seen from the prompt that starts with `ubuntu@`. Check that the toolchain is found:

```
$> riscv32-elf-gcc --version
```

The first line of the output shows the compiler version.

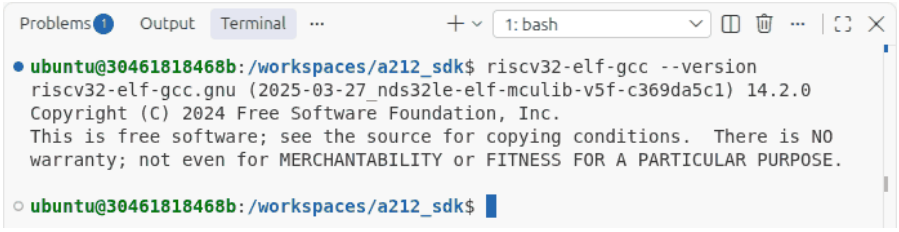


Figure 10: The toolchain version in a VS Code terminal

5.2 Build Everything

- Press `Ctrl+Shift+B`, which runs the default build task **Build all**, or
- run in the terminal:

```
$> make -j$(nproc)
```

All applications are built into the `out/` folder, for example `out/example_service_iq.elf`. Since the SDK folder is shared with the computer, the `out/` folder is also visible outside of the container.

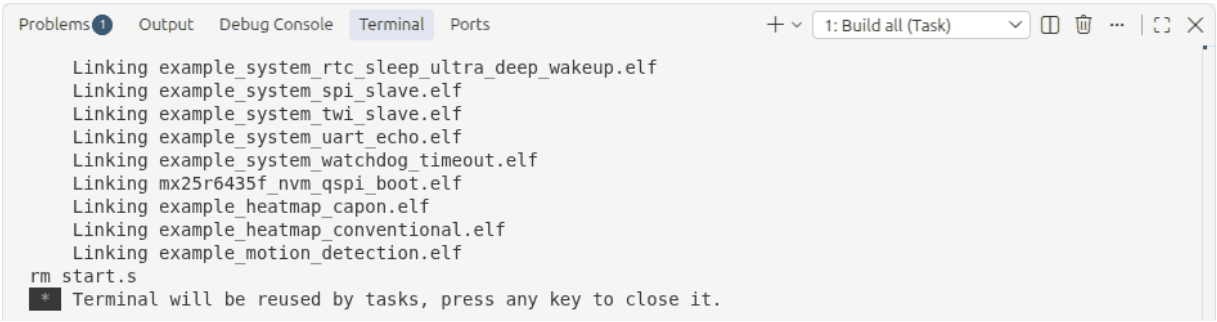


Figure 11: The end of the output from the **Build all** task

Compiler errors and warnings are listed in the **Problems** view (`Ctrl+Shift+M`). Click an entry to jump to the source line.

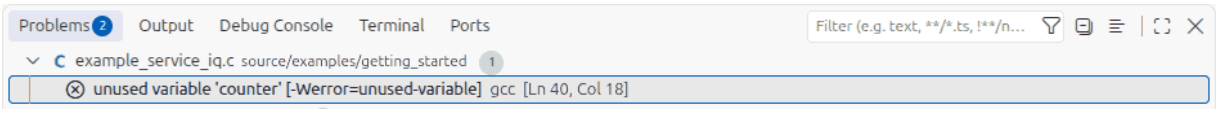


Figure 12: A compiler error in the Problems view

© 2026 by Acconeer AB - All rights reserved

5.3 Build a Single Application

Each application is built from a `.c` file with the same name in the `source/examples/` folders, for example `source/examples/getting_started/example_service_iq.c`.

- Open the application `.c` file in the editor and run the task **Build current file's application**, or
- run the task **Build application...** and enter the application name, or
- run in the terminal:

```
$> make example_service_iq
```

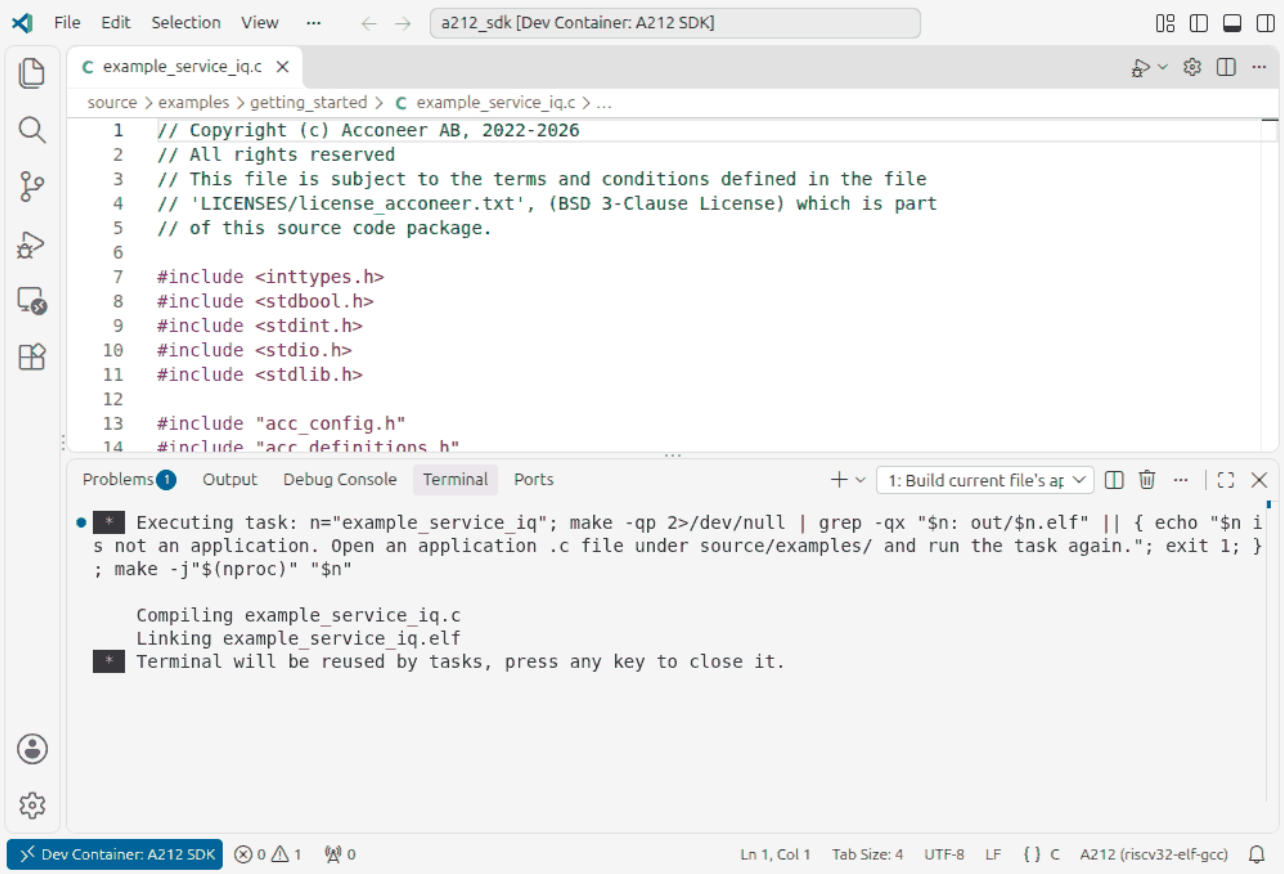


Figure 13: The output from the **Build current file's application** task

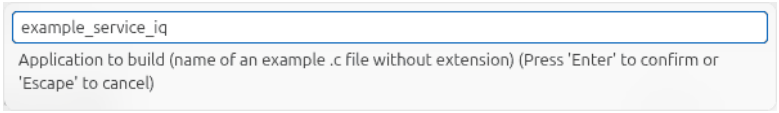


Figure 14: The application name asked for by the **Build application...** task

5.4 Clean

Run the task **Clean**, or:

```
$> make clean
```

5.5 Code Navigation (IntelliSense)

The C/C++ extension is installed in the container automatically and basic code navigation works directly. For the most exact results (correct defines and include paths per file), generate a `compile_commands.json`:

1. Run the task **Generate compile_commands.json**. It rebuilds all applications and records how each file is compiled.
2. The file `compile_commands.json` is created in the SDK root folder and is picked up by the C/C++ extension automatically.

Run the task again after adding new source files.

© 2026 by Acconeer AB - All rights reserved

5.6 Add Custom Code

The `makefile` also picks up files from the following folders, if they exist:

- `user_source/` and `user_include/` for custom `.c` and `.h` files
- `user_lib/` for custom libraries
- `user_rule/` for custom `makefile_define_*.inc` and `makefile_build_*.inc` rules

Use the files in `rule/` (for example `rule/makefile_build_example_service_iq_aps.inc`) as templates for custom build rules.

© 2026 by Acconeer AB - All rights reserved

6 Use the XE212 EVK (XE212 SDK Only)

This chapter applies to the XE212 SDK only. The XE212 SDK adds access to the XE212 EVK to the dev container, so that applications can be loaded and flashed with `xe212tool.py` from VS Code. All commands of `xe212tool.py` are described in the *Using the XE212* chapter of the XE212 SDK User Guide.

The following files are added to the ones in section 1.1 “Overview of the Files”:

File	Purpose
`.devcontainer/devcontainer.json`	Dev container **XE212 SDK**. Works on Linux and Windows
`.devcontainer/linux_usb/devcontainer.json`	Dev container **XE212 SDK (Linux host, USB access)**. Build, load, flash and debug
`.devcontainer/xe212_usb/`	Adds `libusb` and access to the XE212 EVK USB and serial devices to the **XE212 SDK (Linux host, USB access)** container
`.devcontainer/50-xe212.rules`	udev rules that give a normal Linux user access to the XE212 EVK
`.devcontainer/check_buildx.sh`	Checks that the Docker buildx plugin is installed, before **XE212 SDK (Linux host, USB access)** is built
`.devcontainer/prepare_xe212tool.sh`	Downloads the Python packages of `xe212tool.py` when a container is created
`.vscode/tasks.json`	Also contains tasks that load, flash and debug applications on the XE212 EVK
`.vscode/launch.json`	Debug configurations for the VS Code debugger (**Run and Debug**)

Table 3: XE212 dev container files

> **NOTE:** Only one program can access the XE212 USB interface at a time. Close the Acconeer Exploration Tool, and any other program that communicates with the board, before using `xe212tool.py`.

6.1 Choose the Dev Container Configuration

When the SDK folder is opened in a container, VS Code asks which of two configurations to use:

- On Linux, to load or flash the XE212 EVK from VS Code, choose **XE212 SDK (Linux host, USB access)**. Complete section 6.2 “Give the User Access to the XE212 EVK on Linux” first.
- Otherwise, and on Windows, choose **XE212 SDK**.

VS Code remembers the choice. To change it later, see section 7.2 “Leave the Container or Switch Configuration”.

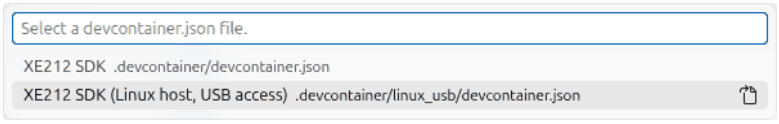


Figure 15: The choice of dev container configuration in the XE212 SDK

XE212 SDK can only build. The built `.elf` files are loaded into the XE212 EVK with `xe212tool.py` running directly on the computer, as described in section 6.5 “Load and Flash on Windows”.

XE212 SDK (Linux host, USB access) can also load and flash applications from inside the container. The container is started with `--privileged` and the host `/dev` folder is shared with the container, so that the XE212 EVK is found even when it reconnects on the USB bus in a different mode. Note that this gives the container full access to all devices of the computer. It also needs the Docker buildx plugin (see section 2.1 “Install Docker Engine”), and stops with a message that says so if it is missing.

6.2 Give the User Access to the XE212 EVK on Linux

1. Copy the udev rules to the system and reload them:

```
$> cd ~/acconeer/xe212_sdk
$> sudo cp .devcontainer/50-xe212.rules /etc/udev/rules.d/
$> sudo udevadm control --reload-rules
$> sudo udevadm trigger
```

© 2026 by Acconeer AB - All rights reserved

2. Add the user to the `dialout` and `plugdev` groups, which are used for serial ports and USB devices:

   ```
   $> sudo usermod -aG dialout,plugdev $USER
   ```

3. **Log out and log in again.**

   The rules give the `plugdev` group access to the EVK, which requires that the group exists on the computer. On distributions without it, the rules still give access to the user logged in at the desktop, but not to users logged in over SSH or to programs started without a desktop session.

4. Connect the XE212 EVK with a USB-C cable and verify that it shows up:

   ```
   $> lsusb -d 0483:
   ```

   A line with ID `0483:a560`, `0483:a561` or `0483:a562` should be printed.

6.3 Load and Flash on Linux

This requires the **XE212 SDK (Linux host, USB access)** configuration and the udev rules from section 6.2 “Give the User Access to the XE212 EVK on Linux”.

If the container was opened with the **XE212 SDK** configuration, switch to **XE212 SDK (Linux host, USB access)** as described in section 7.2 “Leave the Container or Switch Configuration”.

1. Connect the XE212 EVK with a USB-C cable.
2. Check that the EVK is found:

   ```
   $> uv run script/xe212tool.py list
   ```

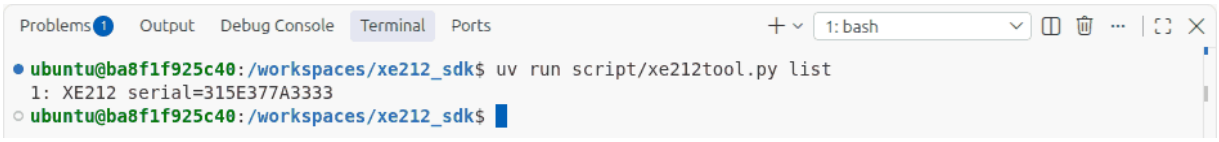


Figure 16: A connected XE212 EVK listed by `xe212tool.py`

3. The first time, make sure that the EVK firmware has the correct version:

   ```
   $> uv run script/xe212tool.py upgrade
   ```

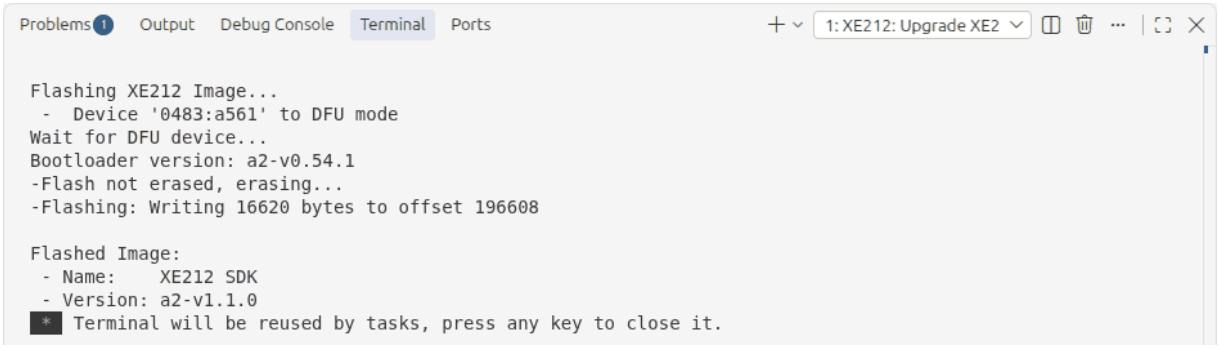


Figure 17: The end of the output from an XE212 EVK firmware upgrade

4. Load an application into the A212 RAM and start it:

   ```
   $> make example_service_iq
   $> uv run script/xe212tool.py dfu out/example_service_iq.elf --log-uart
   ```

   Press `Ctrl+C` to stop printing the log.

5. Or flash the application to the NVM on the EVK, so that it starts at power on:

   ```
   $> make mx25r6435f_nvm_qspi_boot example_service_iq
   $> uv run script/xe212tool.py nvm out/mx25r6435f_nvm_qspi_boot.elf out/example_service_iq.elf --log-uart
   ```

The same commands are available as VS Code tasks. Open the application `.c` file in the editor and run one of the tasks that start with **XE212:**

© 2026 by Acconeer AB - All rights reserved

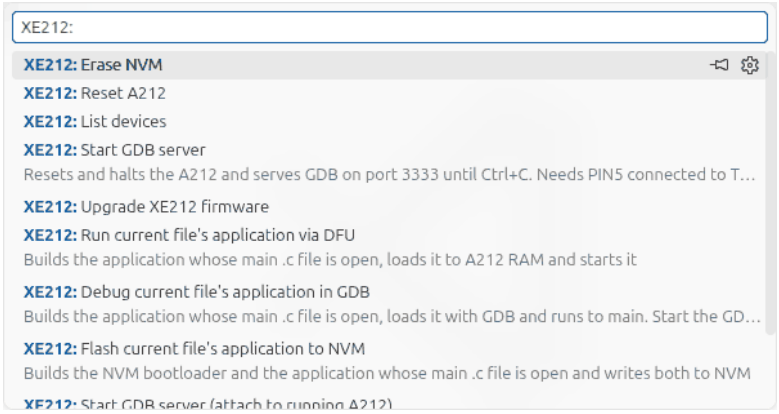


Figure 18: The XE212 tasks in **Terminal > Run Task...**

Task	Description
XE212: List devices	Lists the connected XE212 EVKs
XE212: Upgrade XE212 firmware	Upgrades the XE212 EVK firmware to the version in `bin/`
XE212: Run current file's application via DFU	Builds the open application, loads it to RAM and starts it
XE212: Flash current file's application to NVM	Builds the NVM bootloader and the open application and flashes both
XE212: Reset A212	Restarts the A212, which loads the application from NVM again
XE212: Erase NVM	Erases the application in NVM
XE212: Start GDB server	Resets the A212 and serves GDB, see section 6.4 "Debug with GDB on Linux"
XE212: Start GDB server (attach to running A212)	Serves GDB without resetting the A212, see section 6.4 "Debug with GDB on Linux"
XE212: Debug current file's application in GDB	Builds the open application, loads it with GDB and runs to `main`

Table 4: XE212 VS Code tasks

The **XE212: Run current file's application via DFU** task builds the application, loads it and prints the log from the A212.

© 2026 by Acconeer AB - All rights reserved

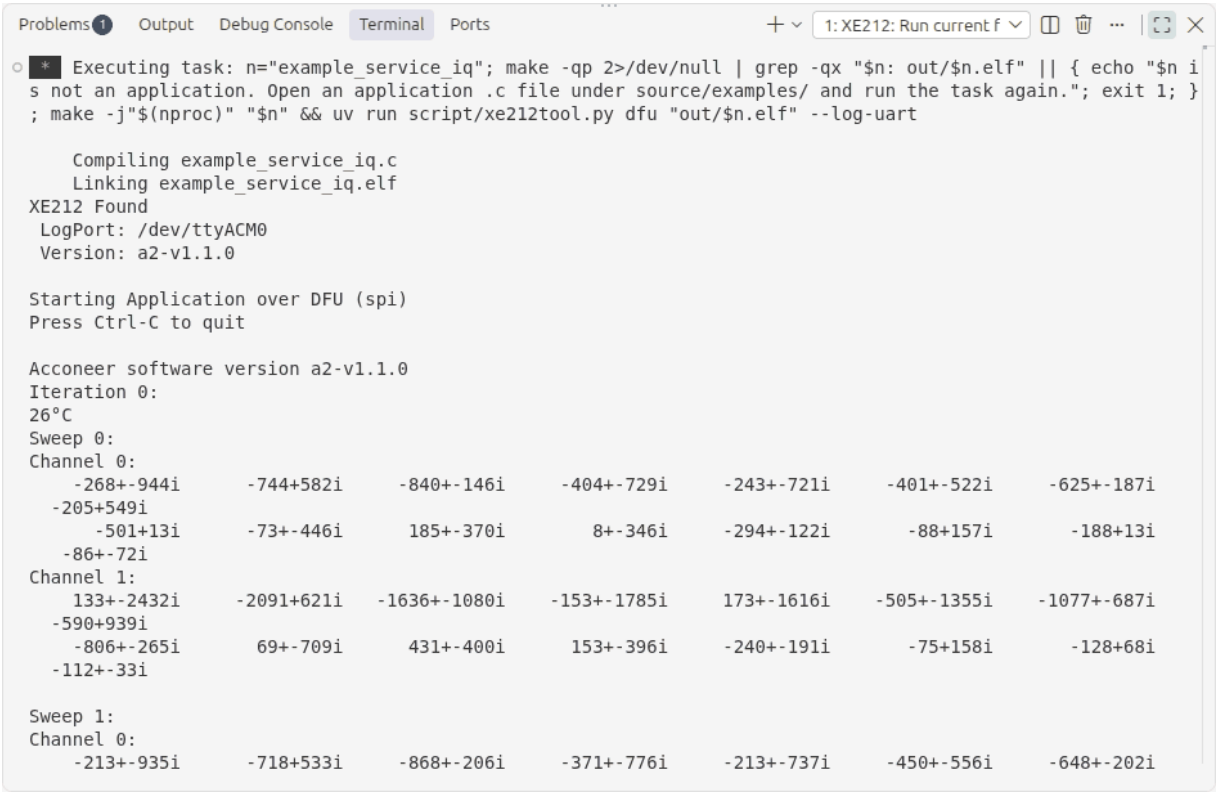


Figure 19: The start of the output from the **XE212: Run current file's application via DFU** task

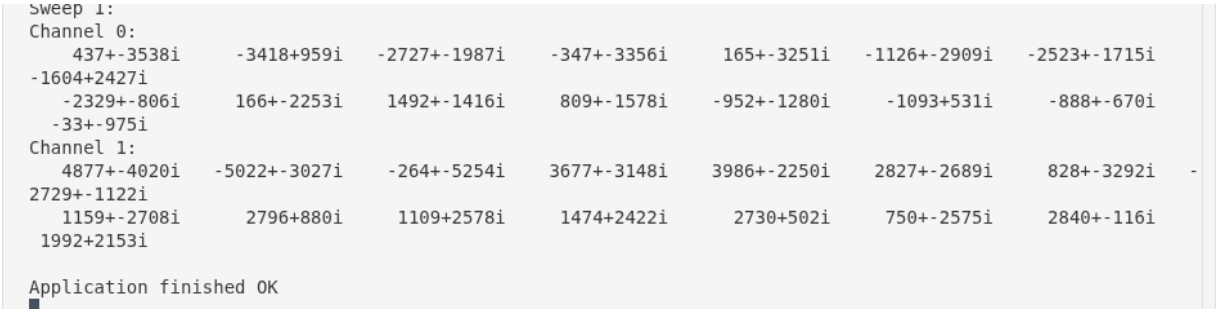


Figure 20: The end of the log from `example_service_iq`

6.4 Debug with GDB on Linux

This requires the **XE212 SDK (Linux host, USB access)** configuration, like section 6.3 "Load and Flash on Linux". The GDB server of `xe212tool.py` debugs the A212 over cJTAG, which needs two wires on the XE212 EVK board. How to connect them, and how to use GDB with the GDB server, is described in the *Debug the A212 with GDB* section of the XE212 SDK User Guide. GDB, `gdb-multiarch`, is included in the dev container.

6.4.1 Debug in VS Code

The VS Code debugger starts the GDB server and GDB by itself. It uses the C/C++ extension, which is installed in the dev container.

1. Open the application `.c` file in the editor.
2. Open **Run and Debug** (`Ctrl+Shift+D`), select one of the debug configurations below and press `F5`:

© 2026 by Acconeer AB - All rights reserved

Configuration	Description
XE212: Debug current file's application	Builds the open application, starts the GDB server, loads the application to RAM, starts it and stops at `main`
XE212: Attach to running application	Starts the GDB server without resetting the A212 and attaches to the application that runs

Table 5: XE212 debug configurations

3. Debug as usual: set breakpoints in the editor with `F9`, step with `F10` and `F11`, continue with `F5`, and look at variables and the call stack in **Run and Debug**. The terminal of the GDB server shows the log from the A212.

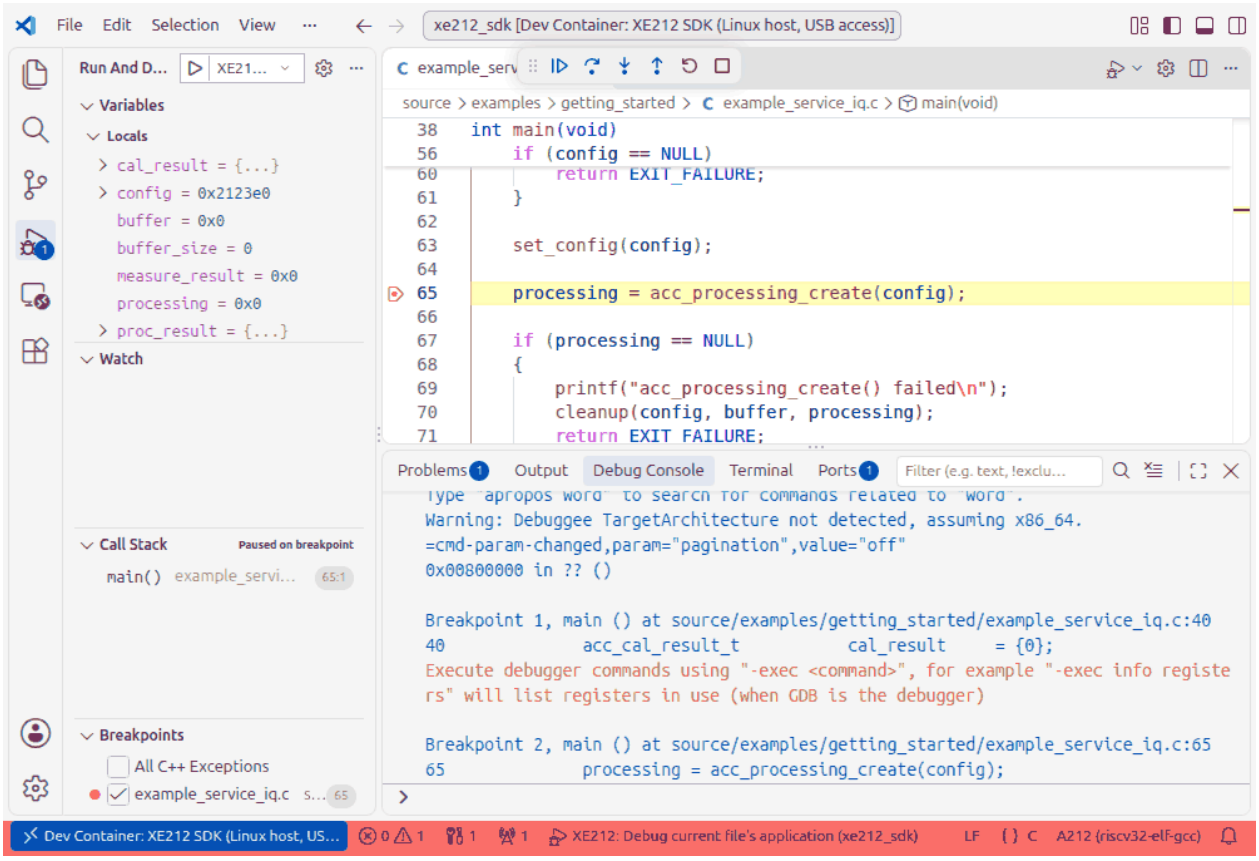


Figure 21: The VS Code debugger stopped at a breakpoint in `example_service_iq.c`

4. Press `Shift+F5` to stop debugging. The application keeps running, and the GDB server is stopped.

The **Debug Console** shows `Warning: Debuggee TargetArchitecture not detected, assuming x86_64` and lines such as `=cmd-param-changed,param="pagination",value="off"`. They are expected: the C/C++ extension does not know the RISC-V architecture of the A212 and shows some GDB messages as they are, but GDB debugs the A212 correctly.

6.4.2 Debug in the Terminal

1. Run the task **XE212: Start GDB server**. It resets and halts the A212, prints the log from the A212 and waits for GDB. To debug an application that is already running, without resetting the A212, run the task **XE212: Start GDB server (attach to running A212)** instead.

© 2026 by Acconeer AB - All rights reserved



Figure 22: The **XE212: Start GDB server** task waiting for GDB

2. Open the application `.c` file in the editor and run the task **XE212: Debug current file's application in GDB**. It builds the application, loads it to the A212 RAM with GDB, starts it and stops at `main`.



Figure 23: GDB stopped at `main` in the **XE212: Debug current file's application in GDB** task

3. Debug with GDB commands in the terminal of the task, for example `next`, `step`, `print` and `continue`. Type `detach` to let the application run without GDB, and `quit` to leave GDB. Run the debug task again to load the application again.
4. Press `Ctrl+C` in the terminal of the GDB server task to stop the GDB server and the log.

The GDB server uses the XE212 USB interface, so stop it before other **XE212:** tasks or debug configurations are run.

6.5 Load and Flash on Windows

On Windows the container cannot reach USB devices. Build in the dev container, and run `xe212tool.py` directly on Windows instead. The built `.elf` files are shared with Windows through the SDK folder.

1. Install `uv` on Windows. Open PowerShell (no administrator rights needed) and run:

   ```
   PS> winget install --id=astral-sh.uv -e
   ```

   Close and reopen PowerShell afterwards so that `uv` is found.

2. Go to the SDK folder in PowerShell:

   ```
   PS> cd \\wsl.localhost\Ubuntu\home\<wsl user>\acconeer\xe212_sdk
   ```

3. Connect the XE212 EVK with a USB-C cable and run `xe212tool.py`:

   ```
   PS> uv run script\xe212tool.py list
   PS> uv run script\xe212tool.py upgrade
   PS> uv run script\xe212tool.py dfu out\example_service_iq.elf --log-uart
   ```

   The first run downloads Python and the required Python packages, which takes a moment.

Build in VS Code (in the container) and load with PowerShell (on Windows). Keep both windows open side by side for a quick edit, build and run cycle.

© 2026 by Acconeer AB - All rights reserved

6.6 Troubleshooting

Symptom	Cause and solution
`xe212tool.py` prints `Could not find any XE212 device`, and `lsusb -d 0483:` on the computer does not list the EVK	The EVK is not connected. Check that the USB-C cable supports data. Some cables only supply power.
`xe212tool.py` in the container prints `Could not find any XE212 device`, but `lsusb -d 0483:` on the computer lists the EVK	The container was not opened with the **XE212 SDK (Linux host, USB access)** configuration (see section 7.2 “Leave the Container or Switch Configuration”), or the udev rules are not installed (see section 6.2 “Give the User Access to the XE212 EVK on Linux”). `ls -l /dev/bus/usb/*/* /dev/ttyACM*` in the container shows whether the device files exist and can be accessed by the user.
`xe212tool.py` fails to connect, or flashing stops halfway	Another program uses the EVK. Close the Acconeer Exploration Tool and other programs that use the EVK, and try again.

Table 6: XE212 EVK problems

© 2026 by Acconeer AB - All rights reserved

7 Tips and Tricks

7.1 Rebuild the Container

The container image is only built the first time. After changing a file in `.devcontainer/`, or after installing a new version of the SDK, press `F1` and run `Dev Containers: Rebuild Container`.

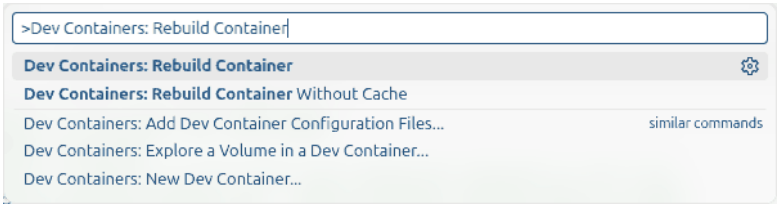


Figure 24: Rebuild Container from the Command Palette

7.2 Leave the Container or Switch Configuration

To leave the container, press `F1` and run `Dev Containers: Reopen Folder Locally` (Linux) or `Dev Containers: Reopen Folder in WSL` (Windows).

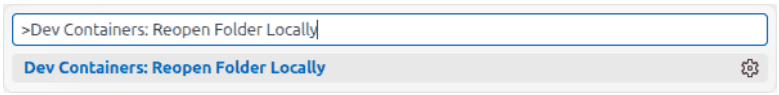


Figure 25: Reopen Folder Locally from the Command Palette

To switch to another dev container configuration, leave the container and run `Dev Containers: Reopen in Container` again. Then select the other configuration.

© 2026 by Acconeer AB - All rights reserved

8 Troubleshooting

Most problems show up as an error message in the VS Code terminal or in the dev container log. To see the log, press `F1` and run `Dev Containers: Show Container Log`.

Symptom	Cause and solution
`make` prints `Make sure to set the NDS32LE_V5F_ROOT`	`make` was run outside of the dev container. Check that the lower left corner of VS Code shows **Dev Container:** and use a terminal opened with **Terminal > New Terminal** in that window.
The notification *An error occurred setting up the container* is shown within a second	This is a generic message. Look in the log, above the line `Stdin closed!`, for the first line that contains `ERROR`, and find it in this table.
The log says `permission denied while trying to connect to the Docker daemon socket`	The user is not in the `docker` group, or has not logged out and in again after being added to it. See section 2.1 “Install Docker Engine”.
The log mentions `buildx` or `BuildKit`, or `docker buildx version` fails	The Docker buildx plugin is missing. Install it with `sudo apt install docker-buildx`.
The log shows an error about a mount, and `snap list docker` lists Docker	Docker installed as a Snap cannot share most folders with a container. Remove it with `sudo snap remove docker` and install Docker Engine as described in section 2.1 “Install Docker Engine”.
The log shows that `docker` cannot be found or started, and VS Code is a Snap or Flatpak	The sandbox of the Snap and Flatpak versions cannot always reach Docker. Install the `.deb` package as described in section 2.2 “Install VS Code”.
On Windows, `make` or `riscv32-elf-gcc` fails with `Segmentation fault`	The `vsyscall=emulate` setting for WSL 2 is missing or not applied. See section 3.1 “Install WSL 2”.
The container image fails to build with a download error	The toolchain is downloaded from GitHub and the base image from Docker Hub. Check the internet connection. If the network requires a proxy, configure it for Docker as described in `https://docs.docker.com/engine/cli/proxy/`.
The log says `sha256sum: WARNING: 1 computed checksum did NOT match`	The downloaded toolchain archive is broken. Press `F1` and run `Dev Containers: Rebuild Container Without Cache` to download it again.
Changes to `.devcontainer/` have no effect	The container image is only built the first time. See section 7.1 “Rebuild the Container”.

Table 7: Dev container problems

© 2026 by Acconeer AB - All rights reserved

9 Disclaimer

The information herein is believed to be correct as of the date issued. Acconeer AB (“Acconeer”) will not be responsible for damages of any nature resulting from the use or reliance upon the information contained herein. Acconeer makes no warranties, expressed or implied, of merchantability or fitness for a particular purpose or course of performance or usage of trade. Therefore, it is the user’s responsibility to thoroughly test the product in their particular application to determine its performance, efficacy and safety. Users should obtain the latest relevant information before placing orders

Unless Acconeer has explicitly designated an individual Acconeer product as meeting the requirement of a particular industry standard, Acconeer is not responsible for any failure to meet such industry standard requirements.

Unless explicitly stated herein this document Acconeer has not performed any regulatory conformity test. It is the user’s responsibility to assure that necessary regulatory conditions are met and approvals have been obtained when using the product. Regardless of whether the product has passed any conformity test, this document does not constitute any regulatory approval of the user’s product or application using Acconeer’s product.

Nothing contained herein is to be considered as permission or a recommendation to infringe any patent or any other intellectual property right. No license, express or implied, to any intellectual property right is granted by Acconeer herein.

Acconeer reserves the right to at any time correct, change, amend, enhance, modify, and improve this document and/or Acconeer products without notice.

This document supersedes and replaces all information supplied prior to the publication hereof.

© 2026 by Acconeer AB - All rights reserved