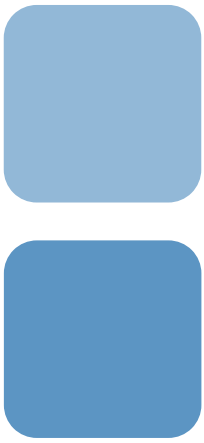
.NET5 on F&S Boards

Version 0.2
(2021-07-23)



Untere Waldplätze 23
D-70569 Stuttgart
Germany

Phone: +49(0)711-123722-0
Fax: +49(0)711-123722-99

About This Document

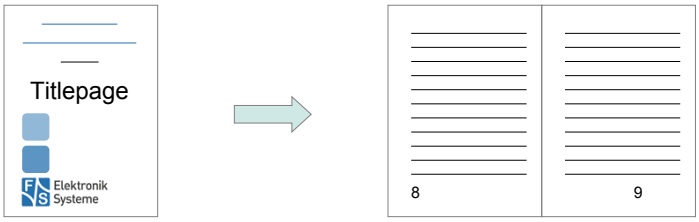
This document describes how to run .NET5 applications on F&S Boards.

Remark

The version number on the title page of this document is the version of the document. It is not related to the version number of any software release! The latest version of this document can always be found at http://www.fs-net.de.

How To Print This Document

This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.



Typographical Conventions

We use different fonts and highlighting to emphasize the context of special terms:

`File names`

Menu entries

`Board input/output`

`Program code`

`PC input/output`

`Listings`

`Generic input/output`

`Variables`

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2021-07-19	0.1	ALL	A	ALL	Initial version	PG
2021-07-23	0.2	ALL	M	ALL	Adapted formatting of document to F&S CI	HF

V	Version
A,M,R	Added, Modified, Removed
Au	Author

Table of Contents

1 Introduction

This document describes how to run .NET5 application to F&S-Linux boards.

Please notice, that there are no native libraries to access Linux hardware peripheries like I2C or SPI in .NET5. If you need to access these in your application, please contact F&S for support to create these libraries.

This document assumes basic knowledge of using Linux on F&S boards. For a detailed introduction please see the *Linux on F&S Boards.pdf* from the document section of your F&S board at

https://www.fs-net.de/

2 System Requirements

The .NET5 images need a lot of disk space, so make sure your flash memory is big enough:

Buildroot

Image type	Image size
Ubifs Image (Nand Flash)	210 MB
Ext4 Image (eMMC)	480 MB

Yocto

Image type	Image size
Ubifs Image (Nand Flash)	
Ext4 Image (eMMC)	

3 Tested Software Versions

The Software used in this document has been tested with the following versions.

Software	Version
Development Machine	F_S_Development_Machine-Fedora_27_V1.2
Buildroot fsimx6	fsimx6-B2020.04
Yocto	-
.NET5	SDK 5.0.302 .NET Runtime 5.0.8
VSCode	1.58.2
VSDebugger	17.0.10413.12

4 Compiling the .Net 5 Images

F&S supports the build environments Buildroot and Yocto to build the system software.

This chapter describes how to build a root file system with preinstalled .NET5 binaries, using Buildroot or Yocto.

For a detailed description how to setup and use the build environments, please see the document *Linux on F&S Boards* chapter *Compiling the System Software*.

4.1 Perquisites

Note

For now, we will only describe how to modify an existing build to add .NET5 support. If there is enough interest in this matter, we will add recipes to build .NET5 images fully automatic.

You can download the .NET5 binaries from the official Microsoft website.

Make sure to download the right OS (Linux) and architecture (Arm32 for i.MX6 and Arm64 for i.MX8 based boards).

If you want to compile the code directly on the board, you will have to download the SDK. This however needs a lot of disc space, so make sure your board has enough flash memory available.

For most cases the .NET5 Runtime should be sufficient.

Copy the Binaries to your development machine.

4.2 Compiling the .NET5 images with Buildroot

1. Get the latest F&S-Buildroot release and execute the `setup-buildroot` script to install Buildroot to your development machine. Follow the instructions.
2. Build the standard defconfig of your machine. Run
   ```
   make fs<YOUR_MACHINE>_std_defconfig
   ```
   in your Buildroot main directory.
3. Open the configuration menu in your buildroot directory
   ```
   make menuconfig
   ```
4. Activate the ICU package at
   `Target packages -> Libraries -> Text and terminal handling -> icu`
5. Build buildroot
   ```
   make -j4
   ```
6. Create the directory
   `output/target/usr/share/dotnet/`
   and copy the previously downloaded .NET5 binaries to it (the complete content of the archive).
7. Create the file
   `output/target/etc/profile.d/dotnet.sh`
   and add the following content:

   ```
   #!/bin/sh
   export PATH=$PATH:/usr/share/dotnet/
   ```

```
export DOTNET_ROOT=/usr/share/dotnet/
```

This will export the path to the .NET5 installation each time you log in.

8. Build buildroot again:
   ```
   make -j4
   ```
9. The build output can be found at
   `output/images/`

4.3 Compiling the .NET5 images with Yocto

TBD

5 Executing .NET applications on F&S boards using .NET5

This chapter describes how to execute .NET5 applications on F&S boards.

5.1 Running an Application

1. Install the .NET5 images to your board. You will need to install kernel, device tree and root filesystem. The different ways of how to install the images are described in the document *Linux on* F&S Boards chapters Image Download and Image Storage.

2. Publish your application as linux-arm for i.MX6/7 boards and linux-arm64 for i.MX8 boards. Use the *–no-self-contained* flag to exclude the runtime binaries from your build

```
dotnet publish -r linux-arm -o bin\linux-arm\publish --no-self-contained
```

3. Boot Linux and transfer your .NET5 application files to the board. You can transfer them via network using the tftp command or use an USB stick. See *Linux on F&S Boards* chapter *Using the Standard System and Devices.*

4. Execute the .NET5 applications DLL using

```
dotnet /path/to/your/application.dll
```

6 Remote debugging .NET5 apps on F&S Boards

You can use Visual Studios Code to program and compile your .NET5 applications as usual, but if you want to debug your application while running on the F&S board, some additional preparations are needed.

6.1 Installing VSDebugger to the board

Download the VSDebugger

For i.MX6/7 from

https://vsdebugger.azureedge.net/vsdbg-17-0-10413-12/vsdbg-linux-arm.tar.gz

For i.MX8 from

https://vsdebugger.azureedge.net/vsdbg-17-0-10413-12/vsdbg-linux-arm64.tar.gz

Note

Remote Debugging was tested with version 17-0-10413-12. There might be a newer version available. You can test it by editing the download string.

The vsdebugger for arm needs about 104 MB of disk space.

If your board has enough flash memory you can install the VSDebugger like the dotnet Runtime binaries:

Buildroot

1. Create the directory `output/target/usr/share/dotnet/vsdbg-linux-arm` and extract the downloaded files to it. (Make sure there is no additional sub directory)
2. Rebuild buildroot and copy the new rootfs to the board.

You can also copy the files to an SD card or USB stick and mount it at the board. You will have to adapt some paths later on then.

6.2 Enabling root access via SSH

VSCode needs root access via SSH for remote debugging.

To allow root to login via SSH with no password set, some preparations are needed.

Please note that this should only be done for development purposes!

Buildroot

1. Mount the rootfile system read-writeable

```
mount -o remount,rw /
```

2. Edit the file /etc/ssh/sshd_conf using the vi editor

```
vi /etc/ssh/sshd_conf
```

3. Edit the following lines (also remove the hashes):

```
(press 'i' to enter edit mode)
#PermitRootLogin prohibit-password -> PermitRootLogin yes
```

```
#PermitEmptyPasswords no          -> PermitEmptyPasswords yes
(press 'Esc' to exit edit mode)
(type ':wq' to save and quit)
```

4. Restart the ssh daemon

```
/etc/init.d/S50sshd restart
```

5. Set an IP address on the board. You can either use DHCP running the command

```
udhcpc
```

or set it per hand with the command

```
ifconfig eth0 up <YOUR.BOARD.IP.ADRESS>
```

You should now be able to log into root per SSH without entering a password.

6.3 Visual Studio Code

6.3.1 Configuring VSCode

In order to launch the VSDebugger on the board, you will have to create or edit the launch.json file. If it does not already exist you will be asked to create it when clicking on the *Run and Debug* tab at the side bar.

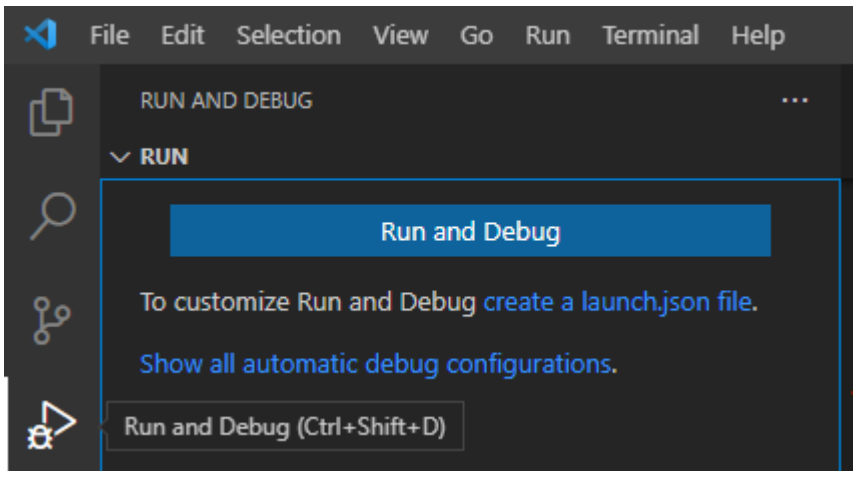


Figure 1 Creating a launch.json file

Replace the configuration with the following.

```
  "configurations": [
    {
      "name": ".NET Core Launch (remote console)",
      "type": "coreclr",
      "request": "launch",
      "preLaunchTask": "FS_Deploy",
      "program": "/usr/share/dotnet/dotnet",
      "args": ["/tmp/${workspaceFolderBasename}/${workspaceFolderBasename}.dll"],
      "cwd": "/tmp/${workspaceFolderBasename}",
      "stopAtEntry": false,
      "console": "integratedTerminal",
      "pipeTransport": {
        "pipeCwd": "${workspaceFolder}",
        "pipeProgram": "C:\\Windows\\System32\\OpenSSH\\ssh.exe",
        "pipeArgs": [
          "root@<X.X.X.X>"
        ],
        "debuggerPath": "/usr/share/dotnet/vsdbg-linux-arm/vsdbg"
      }
    },
  ]
}
```

Edit the following red marked lines if necessary:

- Set your board IP address here. Use ifconfig to show your boards IP address.

```
        "pipeArgs": [
          "root@<X.X.X.X>"
        ],
```

- Change this path if you installed the VSDebugger at a different location

```
      "debuggerPath": "/usr/share/dotnet/vsdbg-linux-arm/vsdbg"
```

- Change these paths if you want to place your application to the flash instead of ram only (e.g to /opt)

```
      "args": ["/tmp/${workspaceFolderBasename}/${workspaceFolderBasename}.dll"],
      "cwd": "/tmp/${workspaceFolderBasename}",
```

- Remove this line if you only want to debug and not rebuild and transfer.

```
      "preLaunchTask": "FS_Deploy",
```

To rebuild and transfer your application each time you start the debugger, you need to create a task.json file Press F1 and type >Task to create it. Copy the following lines to the new task.json:

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "FS_Publish",
            "command": "sh",
            "type": "shell",
            "dependsOn": "build",
            "windows": {
                "command": "cmd",
                "args": [
                    "/c",
                    "\"dotnet publish -r linux-arm -o bin\\linux-arm\\publish --no-self-contained\""
                ],
                "problemMatcher": []
            }
        },
        {
            "label": "FS_Deploy",
            "type": "shell",
            "dependsOn": "FS_Publish",
            "presentation": {
                "reveal": "always",
                "panel": "new"
            },
            "windows": {
                "command": [
                    "C:\\Windows\\System32\\OpenSSH\\ssh.exe",
                    " root@<X.X.X.X>"'mkdir -p /tmp/${workspaceFolderBasename}';",
                    "C:\\Windows\\System32\\OpenSSH\\scp.exe -r",
                    ".\\bin\\linux-arm\\publish\\* root@<X.X.X.X>":/tmp/${workspaceFolderBasename}/"
                ]
            },
            "problemMatcher": []
        }
    ]
}
```

This creates two tasks that will be executed each time before the debugger is started. The first publishes your application for linux-arm without the runtime libraries. The second copies your files via scp to the board. Please note that scp always copies all files to the board. With larger projects it might be useful to only copy the files that have changed. This can be done with tools like rsync. On windows this would be cwrsync, which has to be installed separately.

Edit the following red marked lines if necessary:

- Set your boards ip address

```
" root@<X.X.X.X>"'mkdir -p /tmp/${workspaceFolderBasename}';",
".\\bin\\linuxarm\\publish\\* root@<X.X.X.X>":/tmp/${workspaceFolderBasename}/"
```

- Change these paths if you want to place your application to the flash instead of ram only (e.g to /opt). This has to match the launch.json entry

```
" root@<X.X.X.X>"'mkdir -p /tmp/${workspaceFolderBasename}';",
".\\bin\\linuxarm\\publish\\* root@<X.X.X.X>":/tmp/${workspaceFolderBasename}/"
```

- Change this to linux-arm64 for i.MX8 boards

```
"\"dotnet publish -r linux-arm -o bin\\linux-arm\\publish --no-self-contained\""
```

6.3.2 Start debugging in VSCode

Start debugging by clicking the Run and Debug button at the Run and Debug tab.

Your .NET5 application should be built, transferred to the board. The application should be started and stop, if you have set a break point.

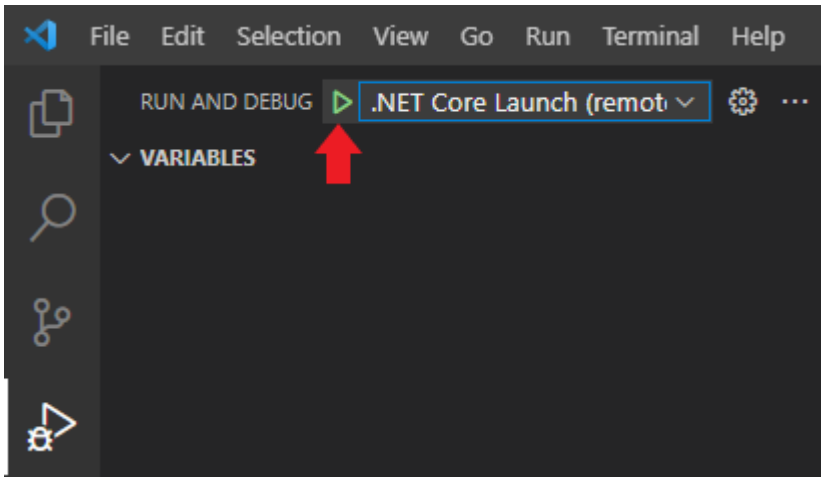


Figure 2 Start Debugging

6.4 Visual Studio 2019

Unfortunately there are no official solutions for remote debugging in Visual Studios 2019 yet.

You could either use Visual Studios Code to debug your application, or try community projects like

https://github.com/radutomy/VSRemoteDebugger

which are not guaranteed to work properly.

7 Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme GmbH (“F&S”) assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.

F&S reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Products are not designed, intended, or authorized for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S product for any such unintended or unauthorized application, the Buyer shall indemnify and hold F&S and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that F&S was negligent regarding the design or manufacture of said product.