Contents

[Introduction 2](#_Toc102499646)

[Debugging using the WebUI 2](#_Toc102499647)

[Log files 2](#_Toc102499648)

[Persistent log files 2](#_Toc102499649)

[Other operations 2](#_Toc102499650)

[Debugging using SSH 2](#_Toc102499651)

[SSH/SCP tools 3](#_Toc102499652)

[Resetting the SSH password 3](#_Toc102499653)

[Logging in using SSH 3](#_Toc102499654)

[Checking the application from the shell 4](#_Toc102499655)

[Running the application manually 4](#_Toc102499656)

[Updating application files using SCP 5](#_Toc102499657)

[Using SSH key authentication 5](#_Toc102499658)

[nidshell for API exploration and debugging 6](#_Toc102499659)

# Introduction

This document describes how applications can be developed and debugged on NordicID’s FR22 device. A basic knowledge of how application development works on the FR22 device is assumed, please see the various samples and documentation provided at <https://github.com/NordicID/fr22_samples/>.

# Debugging using the WebUI

## Log files

Application log data is available in the WebUI by navigating to *Software/Applications* and pressing the *Log* button on the application’s row. The logs for an application can also be reached by navigating to *System/Log* and selecting the application from the *Applications* drop down selector.

A computer screen capture

Description automatically generated with medium confidence

### Persistent log files

Logs are usually only stored in transient RAM memory (i.e., logs are lost when the device reboots or is powered off). By opening the *Download Debug Logs* drop down menu, recording of logs to flash can be enabled. Because logging to persistent storage wears the flash, persistent logging is only enabled for a fixed number of reboots before being disabled automatically.

## Other operations

In addition to the log files, applications can be started, stopped and restarted from the WebUI. If the developer wants to test a new version of the application, the application can also be upgraded via the WebUI (and in this case the system will stop/start the application automatically).

# Debugging using SSH

Secure shell (SSH) can be used to log into the device remotely as the user running the application. Secure Shell Copy (SCP) can also be used to update the application manually.

## SSH/SCP tools

On Linux systems (or Windows systems running Windows Subsystem for Linux (WSL)), the standard ssh/scp clients can be used. Windows also have graphical user interfaces for SSH/SCP available, for example PuTTY (<https://www.putty.org/>) and WinSCP (<https://winscp.net/>).

## Resetting the SSH password

To reset an application’s password, navigate to *Software/Applications* and press the *New Password* button on the application’s row. A dialog will open with the username of the application (using the template ***app-****appname*) and a new temporary random password.

Graphical user interface, text, application, chat or text message

Description automatically generated

Note that this password is a temporary password; it will disappear when the device is rebooted. To get a persistent password for the user, SSH public key authentication should be used (this is described later in this document).

## Logging in using SSH

The password generated in the previous step can now be used to login to the device using SSH and get a command-line shell:

A picture containing graphical user interface

Description automatically generated

### Checking the application from the shell

The running of applications is controlled by the Linux systemd software suite. The command for checking the status of an application from the command-line is *systemctl status* ***nid-app-****appname*, for example with the RfidSample application installed, giving the command *systemctl status nid-app-RfidSample* results in:

Text

Description automatically generated

To view the log of an application in more detail from the commandline, the Linux tool journalctl tool can be used, for example *journalctl -u* ***nid-app-****appname* shows all the available log data for an application.

Graphical user interface, text

Description automatically generated with medium confidence

Adding the -f flag (*journalctl -f -u* ***nid-app-****appname*) only shows the most recent log entries but leaves journalctl running, printing new log entries as they are appended to the log journal by the application.

### Running the application manually

The application can also be started manually on the command-line. To do this, the application must first be stopped in the WebUI (to prevent two instances of the application from running at the same time). After that the application can be started by running the command specified in exec\_start in the application’s manifest.json, e.g. *mono bin/RfidSample.exe:*

Text

Description automatically generated

### Updating application files using SCP

SCP can be used to copy files to the device. This can be used to replace files installed from the application zip file (for example to speed up R&D of the application). This is especially handy if the application is run manually on the command-line, enabling a rapid edit-compile-run R&D cycle without using the WebUI at all. So, assuming we are developing the RfidSample application, it’s already installed, and we are running it from the command-line, we can:

1. Compile the project in Visual Studio
2. Press ctrl-c on the command-line to stop the application
3. Copy the generated build files (exe/dll) to the bin directory on the device, with a tool like WinSCP or as here, with the scp tool:

Text

Description automatically generated

1. Restart the application on the command-line (*mono bin/RfidSample.exe*), immediately running the new version of the application.

(Once the application works as desired, it still needs to be packaged into the application zip as normally done).

## Using SSH key authentication

Using the randomly generated password repeatedly for SSH logins becomes tedious, in addition password logins are disabled after each reboot and the password needs to be regenerated. A more practical method is to use SSH public key authentication.

1. A private/public key pair needs to be generated on the client (e.g., on the application developer’s laptop). This only needs to be done once (the same key pair can be used for all SSH connections). This step depends on the client used; for example, on Linux, the key pair is generated by running the *ssh-keygen* program. If no passphrase is added to the key, the result will be passwordless authentication.
2. The public key is copied to the FR22 device. This can be done by manually copy and pasting the key to the *.ssh/authorized\_keys* file over ssh, or by using tools provided by the ssh client. For example, on Linux clients the ssh-copy-id tool can be used:

Text

Description automatically generated

1. SSH public key authentication is now enabled and is preserved over reboots.

## nidshell for API exploration and debugging

*nidshell* is an interactive API shell available in the FR22 firmware. It can be used to explore and test RPC APIs on the device. It can be started it by entering *nidshell* (followed by <enter>) on an opened ssh connection. *nidshell* supports tab completion; pressing the tab key on the prompt shows the two main level commands: call and exit. exit unsurprisingly causes the shell to exit, but the call command is more versatile; after entering *call* (and pressing the spacebar), the API available on the device can be explored from the command line using the arrow keys.

A screenshot of a computer

Description automatically generated with medium confidence

By navigating down the hierarchy to a partial API topic, the API available under that can also be explored using tab completion, e.g.:

Graphical user interface, text

Description automatically generated

Flags starting with a – sign can be added to the call; the most useful is the -h flag that can be used to retrieve and show the API documentation for a specific topic:

Text

Description automatically generated

Finally, *nidshell* can be used to execute RPC calls from the command-line:

Text

Description automatically generated