Contents

[Introduction 2](#_Toc101802119)

[Prerequisites 2](#_Toc101802120)

[Create the RPC project 2](#_Toc101802121)

[RPC Callbacks 2](#_Toc101802122)

[HTTP API mapping 2](#_Toc101802123)

[Application WebUI 3](#_Toc101802124)

[Create and install the the application 3](#_Toc101802125)

[Accessing the application WebUI 3](#_Toc101802126)

[Notes 4](#_Toc101802127)

# Introduction

This document describes how to create the FR22 RPC WebUI sample application in C#. The source code for the sample can be found at [https://github.com/NordicID/fr22\_samples/tree/master/3. RPCWebUI](https://github.com/NordicID/fr22_samples/tree/master/3.%20RPCWebUI).

The RPC WebUI sample application creates a WebUI that calls the application via the REST API, and the application calls the built-in beeper using the RPC API.

## Prerequisites

* FR22 device with OS version 0.8.0 or later
* fr\_appsigntool files from fr22\_appsigntool directory in <https://github.com/NordicID/fr22_samples/> (requires .NET Core 3.1)
* Visual Studio 2019/2022 with C# support installed
  + It’s possible to use VS Code or the dotnet tool, but this guide assumes Visual Studio is used.
* Knowledge of how the Hello World sample described in 1. FR22 Hello World.docx works
* Knowledge of how the RPC Demo sample described in 2. FR22 RPC demo.docx works

# Create the RPC project

Create a new RPC WebUI project similarly to how the Hello World sample was created and add the NidRpc NuGet as was done in the RPC Demo sample.

# RPC Callbacks

In this sample a nidrpc.Plugin instance is created to provide access to RPC.

\_rpc = new Plugin("application", "RpcWebUI");

*Plugin* extends the *Client* class used in the RPC Demo sample. In addition to the client name, the plugin type (usually *application*) also has to be supplied. In addition to calling other RPC functions, the *Plugin* class provides a way to bind function callbacks to plugin topics, e.g., the following line of code binds the function Beep() to the topic *api/application/RpcWebUI/beep*:

\_rpc["/beep"].CallbackReceived += Beep;

The callback *Beep* gets called with an argument CallbackEventArgs that contains the JSON payload. *Beep* also needs to return a JSON object that will be sent back to the caller.

## HTTP API mapping

RPC calls are also available as HTTP via a REST API. If the required JSON payload for the callback only contains simple key/value pairs, a HTTP GET call can be used using the same path as the callback’s MQTT topic. The HTTP/RPC gateway will transform the HTTP query string to JSON payload for the callback. If a more complex payload is required, HTTP post must be used, with the JSON in the request body. In both cases the JSON returned from the callback will be in the HTTP response body.

# Application WebUI

Static web UI resources (such as html/js/css and image files) of an application are by default retrieved from the frontend directory. This sample has a frontend directory with two files; index.html and RpcWebUI.js. The javascript connects clicking the buttons in index.html into calling the application’s beep API using HTTP POST calls.

In addition, for an application to provide a web UI, it needs to instantiate and connect a NidRpc *Plugin* instance with the plugin type and name as previously described. This is needed so that the HTTP/RPC gateway knows it should handle the mapping and generate a menu entry for the application in the system web UI menu. The web UIs of applications are enabled dynamically at runtime; if an application isn’t running, its’ web UI will not be available.

# Create and install the the application

The application is built, created and installed in the same way as the previous samples. After the application has been installed and started, a new *Application/RpcWebUI* entry will appear in the menu of the device web UI:

Graphical user interface, application

Description automatically generated

## Accessing the application WebUI

The application UI is opened by clicking on *RpcWebUI* in the menu. The sample contains a simple UI with two buttons *Short beep* and *Long beep.* Pressing one of the buttons will beep the built-in buzzer (note, some FRR variants do not have a built-in buzzer, in this case an error alert will be shown instead):

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

Description automatically generated

## Notes

The UI sample beeps the buzzer by first calling its’ own *api/application/RpcWebUI/beep* API, and then the application running on the device accesses the beeper using the built-in *api/builtin/beeper/beep* API. In this case, the application web UI could also have called the built-in *api/builtin/beeper/beep* API directly.