NHS31xx XF Host Apps - README

NXP public
Last modification date: 2022-01-24

NXP cannot provide support relating to XF development.

About

The demo host APPs listed here are developed to highlight the functionality of the corresponding firmware demo applications. These are provided in binary form, ready to install; and with full sources. Use the binary offering to set up a demo immediately; use the sources to learn how communication is possible, how a simple APP could look like, and to jump-start your own APP development efforts.

XF

The technology used for the host APPs is [Xamarin Forms](#), using [Microsoft Visual Studio 2019](#). This is neither a limitation nor an endorsement. To make a mobile or PC application that can communicate with the NTAG SmartSensor, choose whatever suits best for you.

- You can work with any language - C++, Python, Swift, Java, ...
- on any IDE - QT, PyCharm, XCode, Android Studio, ...
- using any library - [Taplinx](#), language specific, using ISO/IEC 7816 commands, using tag specific low-level commands, ...

All demo applications highlight a different use case, but communicate in very similar ways:

- using the NFC interface as a shared memory between two parties,
- exchanging NDEF messages between the host and the ARM microcontroller
- binary encoding commands and responses

When both sides - the firmware and the host code - implement the same set of commands and responses, communication is possible. More information is available in `<SDK>/docs/communication.html`.

Binaries

Use the table below to quickly install the binaries for your platform:

	Android	iOS	macOS	UWP (Win10)
Temperature Logger	Play Store	App Store	Build and run from sources	Sideloading binaries available in the SDK
SensorBoard Monitor	Play Store	AppStore	Build and run from sources	Sideloading binaries available in the SDK
SensorButton Monitor	Play Store	AppStore	Build and run from sources	Sideloading binaries available in the SDK
Signed URL	Play Store	App Store	n.a.	n.a.
Therapy Adherence	Play Store	App Store	n.a.	Sideloading binaries available in the SDK
(one-time) NFC Program Loader	Play Store	Build and run from sources	n.a.	Sideloading binaries available in the SDK

Notes on macOS

- Tested external USB NFC readers are [CLOUD 3700F](#) from Identiv and [ACR122U-A9](#) from ACS.

Notes on Win10

- First install the dependencies, located in the same folder as the APP installer.
    - `Microsoft.NET.CoreFramework.Debug.2.2.appx`
    - `Microsoft.NET.CoreRuntime.2.2.appx`
    - `Microsoft.UI.Xaml.2.4.appx`
    - `Microsoft.VCLibs.x64.14.00.appx`
- Next install the certificate `.cer` file, also located in the same folder as the APP installer. Do this using the options
    - *Store Location*: `Local Machine`
    - *Place all certificates in the following store*: `Trusted People` (click Browse)
- Finally install the application bundle `.msixbundle` or `.appxbundle`, located under the `<SDK>/sw/XF/<APP>/release` subfolder in this SDK.
- Tested external USB NFC readers are [CLOUD 3700F](#) from Identiv and [ACR122U-A9](#) from ACS.

Sources

> If you're new to Xamarin Forms, see
>
> - [Hello World in 10 minutes](#)
> - [Button Tutorial](#)
> - [Data Binding Basics](#)
>
> and other online material.

The provided MS VS 2019 projects can be compiled on 2 different platforms, targeting 4 platforms:

- On Win10, the projects can produce Win10 application bundles and Android packages.
- On macOS, the projects can produce macOS applications, iOS APPs and Android packages.

The sources are complete, safe for the certificates, keys and profiles. Be sure to check and adhere to the license present in the root of the SDK.

Installation

Windows

Requirements

A Windows 10 PC to:

- build Android, iOS, macOS and Windows code.
- generate Android and Windows packages.
- connect via `ssh` to a macOS PC which generates iOS and macOS packages.

Steps

- Install *Microsoft Visual Studio 2019 Professional*

- When installed, launch *Microsoft Visual Studio Installer* from the Start menu.

  - Update the current installation.

  - When updated, modify the current installation:

    - From *Workloads*, select and install *Mobile development with .NET*
    - From *Individual components*, select and install latest of *Windows SDK Version 10.0.xxxxx.x*

- Launch *Microsoft Visual Studio 2019*

  - Select *Tools > Android > Android SDK Manager...*

    - Install *Android 11.0 - R > Android SDK Platform 30*, *Google APIs Intel x86 Atom_64 System Image* and *Google Play Intel x86 Atom_64 System Image*
    - You can uncheck other SDKs

  - Select *Tools > Options > NuGet Packet manager*

    - Add: name `LiveReload`, URL `https://nugetized.blob.core.windows.net/live-reload/index.json`
    - Add: name `Local Packages`, URL `path/to/XF/nuget`

- During first time loading or first time building, if required, Windows 10 OS settings will open where you need to select *Developer mode*.

macOS

Requirements

A macOS 11.6 Big Sur PC or better to:

- build Android, iOS and macOS code.
- generate Android, iOS and macOS packages.

Steps

- Install *Microsoft Visual Studio 2019*

- Launch *Microsoft Visual Studio 2019*

  - Install all extra components when asked for

  - Update the current installation: *APP > Check for Updates...*

  - When updated, modify the current installation:

- - - *Tools > SDK Manager*
    - Install *Android 11.0 - R > Android SDK Platform 30, Google APIs Intel x86 Atom_64 System Image* and *Google Play Intel x86 Atom_64 System Image*
    - You can uncheck other SDKs
  - Select *APP > Preferences... > NuGet > Sources*
    - Add: name `LiveReload`, URL `https://nugetized.blob.core.windows.net/live-reload/index.json`
    - Add: name `Local Packages`, URL `path/to/XF/nuget`

- Launch the APP *Keychain access* and import the iOS certificate (`.cer` file)

- Install the APP *Xcode* - v13.0 or better

- Launch *Xcode*

  - Install all extra components when asked for

  - Install provisioning profiles

    - *APP > Preferences... > Accounts*
    - Add your Apple ID
    - Select your team
    - Click *Download Manual Profiles*
    - Check under `~/Library/MobileDevice/Provisioning Profiles/` the presence of the provisioning profiles.

Troubleshooting

On Windows - UWP

- When UWP builds fail due to the error

```
1   Unable to get MD5 checksum for the key file "
    <myapp>.UWP_TemporaryKey.pfx". Could not find file 'path\to\sw\XF\
    <myapp>\<myapp>.UWP\<myapp>.UWP_TemporaryKey.pfx'.
```

  you may need to create a new key file. Via the GUI: *Project > Properties > Signing > Sign the assembly*

- When UWP builds fail due to the error

```
1   error MSB3325: Cannot import the following key file: <myapp>.pfx.
2   The key file may be password protected.
3   To correct this, try to import the certificate again or
4   manually install the certificate to the Strong Name CSP
5   with the following key container name: VS_KEY_<mykey>
```

  you need to install the existing `.pfx` file using the *Strong Name tool*:

```
1   "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8
    Tools\x64\sn.exe" -i <myapp>.UWP_TemporaryKey.pfx VS_KEY_<mykey>
```

  When a password is asked for, just hit ENTER when the keys are not password protected.
  See also:

  - https://stackoverflow.com/questions/2815366/cannot-import-the-keyfile-blah-pfx-error-the-keyfile-may-be-password-protec

- https://github.com/honzajscz/SnInstallPfx
- When UWP builds fail due to the error

```
1  SDK folder containing 'UAP.props' for 'UAP 10.0.<version>' cannot be
   located. See http://go.microsoft.com/fwlink/?LinkID=798187 for more
   information.
```

  you can either install the missing UWP platform, or change the target platform in the project.

  - The missing version can be installed using the referenced link http://go.microsoft.com/fwlink/?LinkID=798187
  - The target version can be updated in the project files. Via the GUI: *Project > Properties > Application > Targeting*

On macOS - iOS

- In `Assets.xcassets`, a non-transparent icon must be provided for *App Store iOS 1024pt*

- To get rid of the warning *ITMS-90809: Deprecated API Usage* - urging to use *WKWebView* instead of *UIWebView*, add `--optimize=experimental-xforms-product-type` to *Project > Options > iOS Build > Additional mtouch arguments*.

  If that doesn't help - uploads keep being rejected - ensure the deprecated UI is not in the final image, by adding: `--warn-on-type-ref=UIKit.UIWebView -warnaserror:1503`

  It may be needed to add additionally `--optimize=force-rejected-types-removal`
  See also https://devblogs.microsoft.com/xamarin/uiwebview-deprecation-xamarin-forms/

- If building fails with an error about an unknown developer profile:
  - (Re-)download the profiles for your team in *XCode > Preferences*
  - In XF in `Info.plist`, switch to Automatic Provisioning, and then back to Manual again.

NXP cannot provide support relating to XF development.