Using the NFC interface via the XF Ndef library

> NXP public
> Last modification date: 2020-10-28

About

This is a sparse guide to create a new XF project using the NFC interface, and the *Ndef* library supplied in the NHS31xx SDK. This assumes all installation steps outlined in *Setup for XF projects* have been completed.

Create new project

- *File* > *New* > *Project...*
- Filter on *C#*, *All platforms*, and *Mobile*
  Select *Mobile App (Xamarin.Forms)*
- A folder with the given *Project name* will be created in the provided *Location*
- Check the option *Place solution and project in the same directory*
- Select and deselect the available *Platforms*
- Choose a template - see https://docs.microsoft.com/en-us/windows/uwp/design/layout/page-layout
  - Master-Detail
    The Xamarin.Forms MasterDetailPage is a page that manages two related pages of information – a master page that presents items, and a detail page that presents details

about items on the master page. This article explains how to use a MasterDetailPage and navigate between its pages of information.
    - Shell
      Xamarin.Forms Shell includes a URI-based navigation experience that uses routes to navigate to any page in the application, without having to follow a set navigation hierarchy. In addition, it also provides the ability to navigate backwards without having to visit all of the pages on the navigation stack.
    - Tabbed
      The Xamarin.Forms TabbedPage consists of a list of tabs and a larger detail area, with each tab loading content into the detail area.
    - Blank
      No extra sample pages or data will be created.

Configure project

Reference *Ndef* plugin

- [Solution] > *Add* > *Existing project...*
    - Select `<SDK>/sw/XF/Ndef/Ndef.csproj`
- [platform agnostic project] > *Dependencies* > *Add reference...*
    - Check *Ndef*
    - Check *Helpers*
- [platform specific projects] > *Dependencies* > *Add reference...*
    - Check *Ndef*
    - Check *Helpers*

      This is necessary to enable the detection and usage of the platform specific *Ndef* library at run-time. At compile time, the *Ndef* instance is tied to `NETSTANDARD2_0`. Due to the use of the `Lazy` class to hold the `INdef` implementation, the choice of *Ndef* implementation will be taken during execution, at the moment the instance is requested by calling `Plugin.Ndef.CrossNdef.Current`.

Declare NFC usage intention

- [project Android] > *Properties* > *Android Manifest*
    - Check *NFC* under Required permissions
- [project UWP] > *Properties* > *Application* > *Package Manifest...* > *Capabilities*
    - Check *Proximity*
    - Check *Shared User Certificates*
- [project iOS] > Entitlements.plist > *Entitlements*
    - Check *Enable NFC*
    - Switch to *source* and remove *NDEF*
- [project macOS]
    - TODO

Adapt project settings

> UWP only

Needed to retain access to the NFC interface when built with optimizations. Edit the `.UWP.csproj` file manually (i.e. don't know where to find it in the GUI) and change the option `UseDotNetNativeToolchain` for *at least* the release build configurations:

```
    <UseDotNetNativeToolchain>false</UseDotNetNativeToolchain>
```

Resolve version warnings

Check all IDE parser warnings for version conflicts and recommended version updates. Easiest is to open each `.csproj` file in an editor and correct the numbers manually, then save and allow to reload the projects in the IDE.

> Typically, changes made to resolve these warnings can take up to a minute before they are reflected in the IDE.

Add code

Enable dispatching of NFC messages

> Android only

Adding code to [project Android] > `mainActivity.cs`.
Enable message dispatching by appending to or creating `OnCreate`, `OnResume`, `OnPause`, `OnNewIntent`, `OnNewAdapterState`:

```
// append to OnCreate
Xamarin.Forms.MessagingCenter.Send<Xamarin.Forms.Platform.Android.FormsAppCompatActivity>(this, Plugin.Ndef.MessagingCenterMessages.Android.OnCreate);
```

```
// in MainActivity

protected override void OnResume()
{
    base.OnResume();

Xamarin.Forms.MessagingCenter.Send<Xamarin.Forms.Platform.Android.FormsAppCompatActivity>(this, Plugin.Ndef.MessagingCenterMessages.Android.OnResume);
}

protected override void OnPause()
{
    base.OnPause();

Xamarin.Forms.MessagingCenter.Send<Xamarin.Forms.Platform.Android.FormsAppCompatActivity>(this, Plugin.Ndef.MessagingCenterMessages.Android.OnPause);
}

protected override void OnNewIntent(Android.Content.Intent intent)
{
    base.OnNewIntent(intent);
```

```
18
Xamarin.Forms.MessagingCenter.Send<Xamarin.Forms.Platform.Android.FormsAppCompatActivity, Android.Content.Intent>(this, Plugin.Ndef.MessagingCenterMessages.Android.OnNewIntent, intent);
19  }
20
21  public void OnNewAdapterState(int newState)
22  {
23
 Xamarin.Forms.MessagingCenter.Send<Xamarin.Forms.Platform.Android.FormsAppCompatActivity, int>(this, Plugin.Ndef.MessagingCenterMessages.Android.OnNewAdapterState, newState);
24  }
```

Get hold of the NDEF interface

- [platform agnostic project] > *App.xaml.cs*

```
1   public App()
2   {
3       InitializeComponent();
4       Plugin.Ndef.INdef _ndef = Plugin.Ndef.CrossNdef.Current;
5       _ndef.InitTagReaderAsync(OnTagReaderStatus);
6       _ndef.TagConnected += OnTagConnected;
7       _ndef.TagDisconnected += OnTagDisconnected;
8       MainPage = new MainPage();
9   }
10
11  private void OnTagReaderStatus(object sender, Plugin.Ndef.TagReaderStatusChangedEventArgs e)
12  {
13      System.Diagnostics.Debug.WriteLine($"reader: {e.Status.Reader}, R: {e.Status.IsWriteSupported}, W: {e.Status.IsAutoReadSupported}");
14  }
15
16  private void OnTagConnected(object sender, Plugin.Ndef.TagConnectedEventArgs e)
17  {
18      System.Diagnostics.Debug.WriteLine($"e.NdefRecords.Count: {e.NdefRecords.Count}");
19      foreach (NdefLibrary.Ndef.NdefRecord ndefRecord in e.NdefRecords)
20      {
21          if (ndefRecord.TypeNameFormat == NdefLibrary.Ndef.NdefRecord.TypeNameFormatType.Mime)
22          {
23              string mime = System.Text.Encoding.UTF8.GetString(ndefRecord.Type);
24              string payload = ndefRecord.Payload.Length == 0 ? "" : System.BitConverter.ToString(ndefRecord.Payload);
25              System.Diagnostics.Debug.WriteLine($"MIME: {mime} - {payload}");
26          }
27          else /* Assume NfcRtd */
28          {
29              if (System.Text.Encoding.UTF8.GetString(ndefRecord.Type, 0, ndefRecord.Type.Length) == "T")
30              {
31                  int languageLen = ndefRecord.Payload[0];
```

```
                string text =
System.Text.Encoding.UTF8.GetString(ndefRecord.Payload, languageLen + 1,
ndefRecord.Payload.Length - languageLen - 1).Replace("\0", "");
                System.Diagnostics.Debug.WriteLine($"TXT: {text}");
            }
            else /* Assume "U" */
            {
                string text =
System.Text.Encoding.UTF8.GetString(ndefRecord.Payload, 1,
ndefRecord.Payload.Length - 1);
                System.Diagnostics.Debug.WriteLine($"URL:
{System.BitConverter.ToString(ndefRecord.Payload, 0, 1)} {text}");
            }
        }
    }
}

private void OnTagDisconnected(object sender,
Plugin.Ndef.TagDisconnectedEventArgs e)
{
    System.Diagnostics.Debug.WriteLine("OnTagDisconnected");
}
```

Reading and writing

Automatic read on tap

Done by the underlying OS when a tag is reported. See above.

Explicit write followed by read

Make sure the class that responds to a user action subscribes to the `OnTagConnected` event. When tapped, do not block the GUI thread, nor the thread in which the event handler runs.

```
private void OnTagConnected(object sender, Plugin.Ndef.TagConnectedEventArgs
e)
{
    Task.Run(async () => await NdefWriteRead(<bytes to write>, <expected
bytes to read>));
}

private async Task NdefWriteRead(byte[] payload, byte[] expectedResponse)
{
    List<NdefRecord> ndefRecordsToWrite = new List<NdefRecord> {
        new NdefRecord {
            TypeNameFormat = NdefRecord.TypeNameFormatType.Mime,
            Type = System.Text.Encoding.UTF8.GetBytes("n/p"),
            Payload = payload
        }
    };
    (Plugin.Ndef.Status status, List<NdefRecord> ndefRecordsRead) = await
Plugin.Ndef.CrossNdef.Current.WriteReadAsync(ndefRecordsToWrite);
    bool success = (status == Plugin.Ndef.Status.OK)
        && (ndefRecordsRead.Count >= 1)
        &&
(string.Compare(System.BitConverter.ToString(ndefRecordsRead[0].Payload),
System.BitConverter.ToString(expectedResponse)) == 0);
```

```
    <Notify GUI thread to update with the result>
}
```

Notes

- When making changes to the *Ndef* project, always explicitly clean the *Ndef* project before building your application. A missing *Ndef* library is detected automatically, but not an out-of-date version.

Android

- It is recommended to also implement `OnRequestPermissionsResult` and `AdapterStateActionBroadcastReceiver`. This will notify the application to learn of the user's consent of accessing the NFC interface, and the enabling and disabling of the NFC interface while the application is running, respectively.

```
// append to OnCreate
RegisterReceiver(new AdapterStateActionBroadcastReceiver(), new
Android.Content.IntentFilter(Android.Nfc.NfcAdapter.ActionAdapterStateChanged
));
```

```
// in MainActivity
public override void OnRequestPermissionsResult(int requestCode, string[]
permissions, [GeneratedEnum] Android.Content.PM.Permission[] grantResults)
{
    Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode,
permissions, grantResults);
    base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
}
```

```
// below MainActivity
[Android.Content.BroadcastReceiver]
public class AdapterStateActionBroadcastReceiver :
Android.Content.BroadcastReceiver
{
    public override void OnReceive(Android.Content.Context context,
Android.Content.Intent intent)
    {
        (context as
MainActivity).OnNewAdapterState(intent.Extras.GetInt(Android.Nfc.NfcAdapter.E
xtraAdapterState));
    }
}
```

- It is helpful to set up catch-all handlers to aid in debugging:

```
// append to OnCreate
AppDomain.CurrentDomain.UnhandledException +=
AppDomain_CurrentDomain_UnhandledException;
System.Threading.Tasks.TaskScheduler.UnobservedTaskException +=
TaskScheduler_UnobservedTaskException;
AndroidEnvironment.UnhandledExceptionRaiser +=
AndroidEnvironment_UnhandledExceptionRaiser;
```

```
// in MainActivity

static private void AppDomain_CurrentDomain_UnhandledException(object
sender, UnhandledExceptionEventArgs e)
{

 System.Diagnostics.Debug.WriteLine("AppDomain.CurrentDomain.UnhandledExcept
ion: " + (e.ExceptionObject as Exception).Message);
}

static private void TaskScheduler_UnobservedTaskException(object sender,
System.Threading.Tasks.UnobservedTaskExceptionEventArgs e)
{

 System.Diagnostics.Debug.WriteLine("TaskScheduler.UnobservedTaskException:
" + e.Exception.Message);
}

static private void AndroidEnvironment_UnhandledExceptionRaiser(object
sender, RaiseThrowableEventArgs e)
{

 System.Diagnostics.Debug.WriteLine("AndroidEnvironment.UnhandledExceptionRa
iser: " + e.Exception.Message);
}
```

Windows

- When building for Windows, make sure you target the same Windows version in your *Ndef* project as in your application project.