Use NFC tags with Windows Phone 8

This article explains how to use Proximity API to manipulate NFC tags.

24 Mar 2013

Introduction

NFC is the acronym for **Near Field Communication**; it's a wireless communication technology via radio-frequency which allows the exchange of data between a reader and a target or an NFC terminal at a distance of a few centimeters. Applications include contact-less transactions, data exchange, and simplified setup of more complex communications such as Wi-Fi or Bluetooth. Communication is also possible between an NFC device and an unpowered NFC chip, called a **tag** (a business card, for example).

NFC standards cover communications protocols and data exchange formats, and are based on existing radio-frequency identification (**RFID**) standards. The standards include ISO/IEC 18092 and those defined by the NFC Forum, which was founded in 2004 by Nokia itself, Philips and Sony, and now has more than 160 members.

Like NFC will become more and more present in your life, Windows Phone 8 SDK added NFC support with the Proximity API. This article covers the manipulation of NFC tags with said API.

Tip: This API is a part of WinPRT API; you can use it from both managed and native code. To instantiate a delegate with native code, read Delegates (C++/CX)

Further down the article we will also cover how to "Write a NFC tag using Windows Phone 8".

NFC Chipsets

NFC tags use RFID passive chipset. This chipset can be produced with different technology and available memory. To help you to choose a NFC tag type, this table overview NFC Tags type with their principal characteristics.

Name	Available memory	URL length	Text length	NFC Class	Windows Phone 8 support
Ultralight (UL)	48 bytes	41 characters	39 characters	Type 2	yes (?)
Ultralight C (ULC)	148 bytes	132 characters	130 characters	Type 2	yes (?)
NTAG203	144 bytes	132 characters	130 characters	Type 2	yes (?)
MiFare 1K	1024 bytes	256 characters	709 characters	Classic (1)	yes
Fudan F08	1024 bytes				?
DESFire EV1 2k	2048 bytes			Type 4	yes (?)
DESFire EV1 4k	4096 bytes			Type 4	yes (?)
DESFire EV1 8k	8192 bytes			Type 4	yes (?)
Sony FeliCa 4K	4096 bytes			Type 3	yes (?)
Sony FeliCa Lite	224 bytes			Type 3	yes (?)
Topaz	512 bytes			Type 1	yes

Unfortunately, I could test only MiFare 1Kb and Topaz 512 bytes type and I do not know which tags are supported by Windows Phone. You can find more information here: Nfc Tag.

Windows Phone 8 supports only "NDEF" formatted tags. NDEF are NFC Data Exchange Format (NDEF) messages. You can find more information here: Understanding NFC Data Exchange Format (NDEF) messages. To buy NFC tags, you can find a list of retailers on XDA Wiki.

If your tag is not formatted you can use a NFC writer or an Android device + NFC TagWriter by NXP application.

Warning: Nexus 4 and Nexus 10 don't support MiFare 1K tags.

(?) These tags are normally compatible with Windows Phone. Confirmation pending.

Limitations with Windows Phone 8

(C) Copyright Nokia 2014. All rights reserved.

Since Proximity API gives only high level access to the NFC protocol and Windows Phone adds some protection on top of that, your interaction with NFC tags is limited:

- You cannot enable the write lock. How can I lock / write protect NFC Tags?]
- You cannot format a tag. Your tag must be formatted for NDEF message.
- Your tag can contain only an NDEF message.
- You cannot use all tag memory. NDEF formatting data and other private tag data take memory. Because of the fact that a tag has only a few bytes of memory, this data size is important. For example, you can only write 716 bytes on a tag with a memory of 1024 bytes and 454 bytes on a tag with a memory of 512 bytes.
- Proximity API does not give tools to manipulate directly NDEF messages. To manipulate raw NDEF messages you can use NDEF Library for Proximity APIs (NFC).
- You cannot receive or write a tag when your application runs in background.

Use Windows Phone 8 Proximity API

It is possible with Windows phone 8 to write on NFC tags. NFC tags can take several forms such as stickers, key rings, cards or phone stand like Nokia Wireless Charging Stand (DT-910).

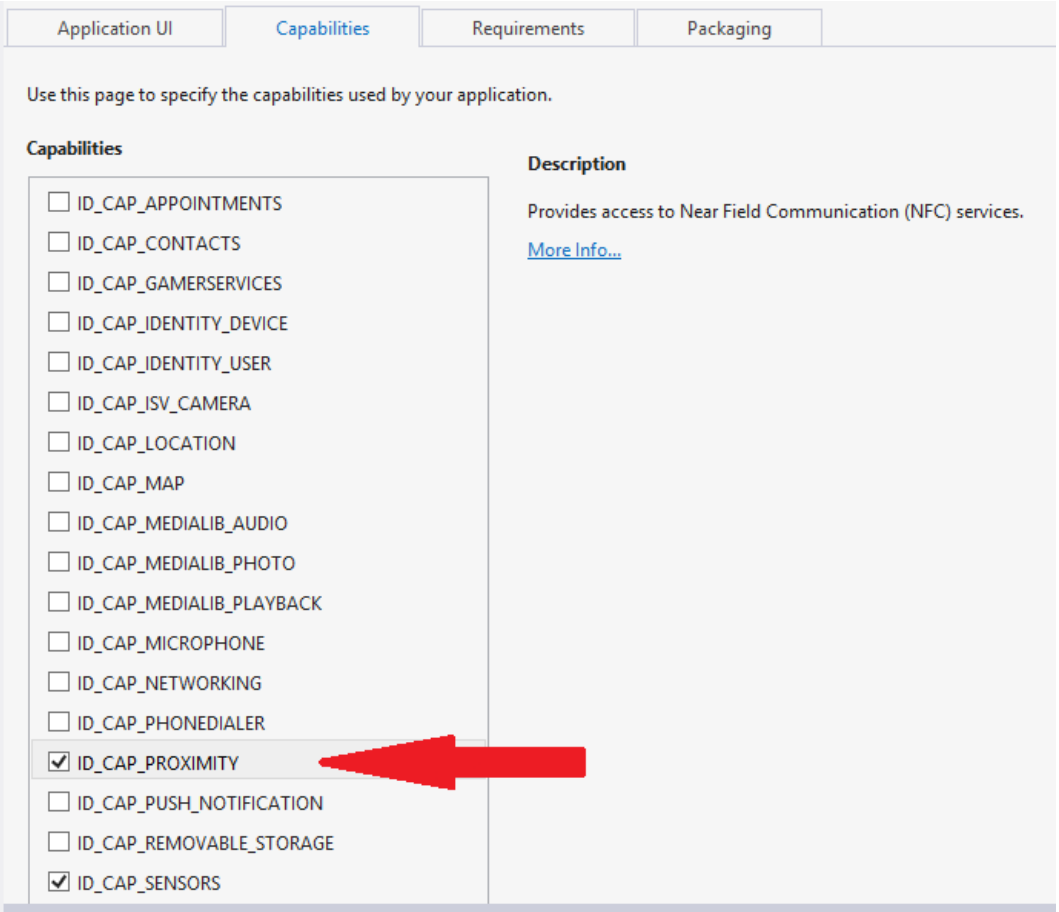
To do this, we will create an application that will:

- detect the presence of a writable nfc tag
- write data to the tag

Create Project

NFC communications are provided by `ProximityDevice` class of `Windows.Networking.Proximity` namespace. To use NFC, you need to add a specific Capabilities to **WMAppManifest.xml** file.

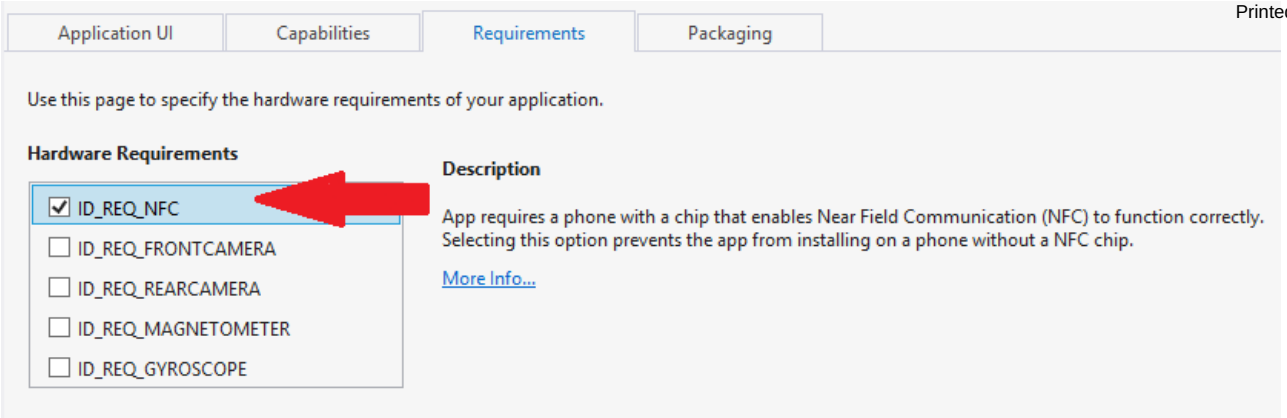
- Open your **WMAppManifest.xml** and add the capability `ID_CAP_PROXIMITY`.



If you don't do this, an exception will be launched when you will try to access to `ProximityDevice`.

- If NFC is a requirement, you can limit your application to Windows Phone device with NFC capability. edit **Properties -> WMAppManifest.xml** file and check **ID_REQ_NFC** in **Requirements** tab.

(C) Copyright Nokia 2014. All rights reserved.



Test if NFC is available

NFC chipset aren't mandatory on windows phone, so before use NFC, it's necessary to test if NFC chipset is present. To read and write a NFC tag you must use the ProximityDevice class. To access this object you must use the static function ProximityDevice.GetDefault(). If this method returns `null`, your device does not support NFC.

```
if (ProximityDevice.GetDefault() != null)
  MessageBox.Show("NFC present");
else
  MessageBox.Show("Your phone has no NFC or NFC is disabled");
```

Once you have access to ProximityDevice, you can use two interesting events to be notified when a NFC communication is available:

- DeviceArrived : Occurs when the device enters in contact with a device/tag.
- DeviceDeparted : Occurs when the device loses the contact with a device/tag.

Subscribe to read a NFC Tag

To receive data you must register a callback handler with the SubscribeForMessage method:

```
long subscribedMessageId = device.SubscribeForMessage("Windows.SampleMessage",
messageReceivedHandler);

void messageReceivedHandler(ProximityDevice device, ProximityMessage message)
{
    // Write message parsing code here...
}
```

This method takes two parameters:

- **protocol**: selects which NFC message you want to receive.
- **handler**: used to receive data.

If successful, an ID is returned. This ID is used to unsubscribe your handler with the StopSubscribingForMessage method:

```
device.StopSubscribingForMessage(subscribedMessageId);
```

When a message is received, binary data is accessible by the ProximityMessage handler parameter.

Warning: To receive a message, you must perform your subscription before the ProximityDevice.DeviceArrived event is fired, or user must remove the tag and repeat the operation.

Publish to write a NFC Tag

To write a tag, you must use the PublishBinaryMessage method.

(C) Copyright Nokia 2014. All rights reserved.

```
long publishedMessageId=   proximityDevice.PublishBinaryMessage(   "LaunchApp:WriteTag",
buffer, messageWrittenHandler);

void messageWrittenHandler (ProximityDevice sender,  long publishedMessageId)
{
    // The LaunchApp message has been successfully written to a tag.
}
```

Parameters are :

- **protocol**: selects which NFC message you want to write.
- **data**: binary data
- **handler** (optional): used to know if data has been written. Second parameter is the publisher ID given by `PublishBinaryMessage` method.

Like `SubscribeForMessage`, this method returns an ID used to stop publication. To stop the publication, your must use ProximityDevice.StopPublishingMessage.

```
void messageWrittenHandler (ProximityDevice sender,  long publishedMessageId)
{
//message have been written, stop publishing
    proximityDevice.StopPublishingMessage(publishedMessageId);
}
```

Tip: Unlike the "read" action, if your device is already in contact with the tag, the message will be written anyway.

A simple Publish and Subscribe example of NFC communication: File:NFC WP8 sample app.zip

Detect presence of writable nfc tags

We will first try to detect that a writable tag is here for instance to display a message.

```
ProximityDevice.GetDefault().SubscribeForMessage("WriteableTag", WriteableTagDetected);
private void WriteableTagDetected(ProximityDevice sender, ProximityMessage message)
{
 Deployment.Current.Dispatcher.BeginInvoke(() =>
 {
  MessageBox.Show("message");
 });
}
```

Write content to nfc tag

The previous step is not necessary, but it can be interesting to give a feedback to the user.

To write to a writable tag, just publish a message of the following type:

Protocol	Description
Windows:WriteTag	Same as the Windows protocol, except that the content is intended to be written to a static tag. The message is not transmitted to any device except a writable static tag. This protocol is only valid for publications.
WindowsUri:WriteTag	Same as the WindowsUri protocol, except that the content is intended to be written to a static tag. The message is not transmitted to any device except a writable static tag. This protocol is only valid for publications. No subtype.
LaunchApp:WriteTag	Write a tag that can be used to launch a specific app with optional launch parameters. If you publish a LaunchApp:WriteTag message to a tag the result when that tag is tapped to a computer is the same as

(C) Copyright Nokia 2014. All rights reserved.

	calling the PeerFinder.Start method and passing your launch parameters.

Protocol

When you want to publish or subscribe to messages, you need to specify the type of messages you want and supported protocol. Message type values are case-sensitive strings consisted of one part or two parts:

- <protocol>.<subtype>
- <protocol>

This part will explain these protocols with different use case.

Protocol	Description
Windows	The message contains binary data.
WindowsUri	The message data contains an URI (UTF-16LE encoded string). No subtype.
WindowsMime	The message data is of a specific MIME type. For example, if the message data is a JPEG image, the message type is `WindowsMime.image/jpeg`.
Windows:WriteTag	Same as the `Windows` protocol, except that the content is intended to be written to a static tag. The message is not transmitted to any device except a writable static tag. This protocol is only valid for publications.
WindowsUri:WriteTag	Same as the `WindowsUri` protocol, except that the content is intended to be written to a static tag. The message is not transmitted to any device except a writable static tag. This protocol is only valid for publications. No subtype.
LaunchApp:WriteTag	Write a tag that can be used to launch a specific app with optional launch parameters. If you publish a `LaunchApp:WriteTag` message to a tag the result when that tag is tapped to a computer is the same as calling the `PeerFinder.Start` method and passing your launch parameters.
WriteableTag	When subscribing for this message protocol, if a writable tag is brought in to proximity, a proximity message is received that contains an `int32` (little endian) indicating the maximum writable size of the tag. This protocol is only valid for subscriptions.
NDEF	The message contents are properly formatted NDEF records. The underlying type of the content for a publication using NDEF as the message type is contained in the NDEF records. A subscription for the NDEF type subscribes to all NDEF formatted messages.
NDEF:ext	The message data is application defined NDEF records (TNF field value of 0x04). This protocol only applies to subscriptions; when publishing NDEF content, use NDEF.
NDEF:MIME	The message data is a properly formatted NDEF mime message (TNF field value of 0x02). For example,`NDEF:MIME.image/jpeg`. This protocol only applies to subscriptions; when publishing NDEF content, use NDEF.
NDEF:URI	The message data is a properly formatted NDEF message of a type defined by a URI namespace (TNF field value of 0x03).This protocol only applies to subscriptions; when publishing NDEF content, use NDEF. This means that the data format is identified by the specified URI.
NDEF:wkt	The message data is a properly formatted NDEF message of a type defined by the NFC forum (TNF field value of 0x01).An example of this type is`NDEF:wkt.U` for the well known type of URI. This protocol only applies to subscriptions; when publishing NDEF content, use NDEF.
NDEF:WriteTag	The message data should be written to an NFC forum standard static tag. The message data must be in a proper NDEF format. This protocol is only valid for publications.
NDEF:Unknown	The message data is an untyped NDEF message (TNF field value of 0x05).This protocol only applies to subscriptions; when publishing NDEF content, use NDEF.

The subtype is a string of alphanumeric characters and any of the valid URI characters as defined by RFC 3986: **- . _~ : / ? # [ ] @ ! $ & ' ( ) * + , ; = %**. The subtype cannot exceed a length of 250 characters.

To write a NFC tag you generally contact selected protocol name with **:WriteTag**.

(C) Copyright Nokia 2014. All rights reserved.

WriteableTag

This protocol is only available for subscription. When your device gets in contact with a writeable NFC tag, the registered handle receives an `int32` value. This value is the real tag's writeable capacity.

```
device.SubscribeForMessage("WriteableTag", WriteableTagHandler);

private void WriteableTagHandler(ProximityDevice sender, ProximityMessage message)
{
    //convert binary to int32
    var WritableSize = System.BitConverter.ToInt32(message.Data.ToArray(), 0);
 }
```

Windows.<subtype>

Windows protocol is used to define a custom type to write custom binary data. It is a high level protocol which hides NDEF manipulation. To use it you must specify your custom subtype:

```
//write byteBuffer to the tag
proximityDevice.PublishBinaryMessage(  "Windows:WriteTag.mySubType", byteBuffer,
messageWrittenHandler);

//subscribe
device.SubscribeForMessage("Windows.mySubType", messageReceivedHandler);
...
void messageReceivedHandler(ProximityDevice device, ProximityMessage message)
{
  // get binary data received
  var byteBuffer = message.Data;
}
```

Note: The subtype is a string of alphanumeric characters and any of the valid URI characters as defined by RFC 3986: **- . _ ~ : / ? # [ ] @ ! $ & ‘ ( ) * + , ; = %**. This string cannot exceed a length of 250 characters.

WindowsUri

WindowsUri protocol is used to read and write URI. It is a high level protocol which interfaces a NDEF URI message.

Note: URI is encoded in UTF16-LE.

When you receive an URI, you must convert binary data to string with the UTF16-LE decoder.

```
//subscribe
device.SubscribeForMessage("WindowsUri", messageReceivedHandler);
...
void messageReceivedHandler(ProximityDevice device, ProximityMessage message)
{
  // get binary data received
   var buffer = message.Data.ToArray();
   var sUri = Encoding.Unicode.GetString(buffer, 0, buffer.Length);

  //remove null character if present
   if (sUri[sUri.Length - 1] == '\0')
       sUri = sUri.Remove(sUri.Length - 1);
}
```


(C) Copyright Nokia 2014. All rights reserved.

Warning: Windows Phone will always handle "WindowsUri" messages by prompting the user.

When you write an URI, you must convert your string to binary with an UTF16LE encoder.

```
var uri = new Uri(" http://www.nokia.com/");
//encode uri to UTF16LE.
var buffer = Encoding.Unicode.GetBytes(uri.ToString());
publishId = device.PublishBinaryMessage("WindowsUri:WriteTag", buffer.AsBuffer(),
publishHandler);
```

A NFC tag which contains an URI can perform interesting interactions with Windows Phone:
You can find more details in the below section - "Write a NFC tag using Windows Phone 8", for now the below table covers all interactions in a quick go.

Action	URI format
Open a web page	URI is an URL. For example http://www.nokia.com/
URI associations	Custom URI can launch a specific application. You can read more information here URI associations for Windows Phone
Geo tag	http://m.ovi.me/?c=<latitude>,<longitude> [How to Store Geo Coordinates on an NFC Tag]
Launch a driving to a position	ms-drive-to:?destination.latitude=<latitude>&destination.longitude=<longitude> How to request driving or walking directions
Dial a phone number	tel:+*33600000*
Send a mail	mailto:*doctor@tardis.time* You can add other information like a subject, a body.. For more information see here
Use Nokia MixRadio	Nokia MixRadio app-to-app protocol for Windows Phone 8
Launching built-in apps and WP8 settings	URI schemes for launching built-in apps for Windows Phone 8
Dynamic Tag	To generate a Dynamic Tag, you can use HTTP URI which perform an URI Redirection. Rediction can return a new URI or a file like a vcard, image,... For example read Web Redirection Script
...	

WindowsMime.(mimetype)

WindowsMime protocol is used to read and write binary data identified by a MIME type. It is a high level protocol which interfaces a NDEF Mime message. The MIME type is optional.

If you don't specify a MIME type, the first 256 bytes contain the MIME type in ASCII terminated by '\0". You must read it to know the format of the following data.

```
device.SubscribeForMessage("WindowsMime", WindowsMimeHandler);

private void WindowsMimeHandler(ProximityDevice sender, ProximityMessage message)
{
        var buffer = message.Data.ToArray();
        int mimesize =0;
       //search first '\0' charactere
       for(mimesize = 0;   mimesize <256 && buffer[mimesize] !=0;  ++mimesize)
       {
        };

       //extract mimetype
       var mimeType = Encoding.UTF8.GetString(buffer, 0, mimesize);
```


(C) Copyright Nokia 2014. All rights reserved.

```
        //convert data to string. This traitement depend on mimetype value.
        var data = Encoding.UTF8.GetString(buffer, 256, buffer.Length - 256);
}
```

If you specify the MIME type, your handle will be called for selected NDEF Mime message and the first 256 bytes will be removed. You get directly the binary data.

```
device.SubscribeForMessage("WindowsMime.text/plain", WindowsMimeHandler);

private void WindowsMimeHandler(ProximityDevice sender, ProximityMessage message)
{
    //receive text/plain data. Convert it to string
     var data = Encoding.UTF8.GetString(buffer, 256, buffer.Length - 256);
}
```

Warning: Windows Phone will always handle "WindowsUri" messages by prompting the user.

To write a tag, you must specify the MIME type and you must convert your data to binary correctly.

```
var buffer = Encoding.UTF8.GetBytes("Who are you?" );
publishId = device.PublishBinaryMessage("WindowsMime:WriteTag.text/plain",
buffer.AsBuffer(), publishHandler);
```

Write a MIME message to a tag adds a few interesting interactions:

Action	Mime type	Desciption
small picture	▪ image/jpeg ▪ image/gif ▪ image/png	Binary data
contact infomation	text/x-vcard	VCard data. VCard specification can be found here
...		

LaunchApp

This protocol is only available to write a tag. This tag is used to launch a Windows 8 and/or Windows Phone 8 application with arguments. To launch a Windows Phone application, the command is decomposed in three part concatenated by a tab char:

- Argument: argument passed to the application.
- **WindowsPhone**: platform type. It could be **Windows** or
- ApplicationId : Windows Phone Application ID.

For example

```
Time_lord=Doctor_who\tWindowsPhone\t{8ac6c849-7b2f-4fa4-9be4-7e9d5f3e46a2}
```

Note: To write the tag correctly, you must convert command line to binary with an UTF16-LE encoder.

```
 //create a command with launch this application with an argument.
 string command ="text=" +NfcMessage.Text+ "\t";
command +="WindowsPhone" + "\t";
command += "{" + Windows.ApplicationModel.Store.CurrentApp.AppId.ToString() + "}";
```


(C) Copyright Nokia 2014. All rights reserved.

```
var byteArray = Encoding.Unicode.GetBytes(command);
publishId = device.PublishBinaryMessage("LaunchApp:WriteTag", byteArray.AsBuffer(),
publishHandler);
```

When your application is launched from an NFC tag, you can get the argument in `OnNavigationTo` method with the **ms_nfp_launchargs** key.

```
 protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
            base.OnNavigatedTo(e);
            string parameter = string.Empty;
            if (NavigationContext.QueryString.TryGetValue("ms_nfp_launchargs", out
parameter))
            {
                MessageBox.Show("Congratulation\nYou launch application with a NFC
tag.\nParamaters : "+ parameter);
                NavigationContext.QueryString.Remove("ms_nfp_launchargs");
             }
}
```

Tip: If the application is not installed, Windows Phone will ask to install it.

NDEF

NDEF protocol is used to manipulate NDEF messages by yourself. When you receive a message, you get the binary NDEF format and when you publish data, you must create a binary NDEF Message. You can use the third party library NDEF Library for Proximity APIs (NFC) for this.

```
//read
device.SubscribeForMessage("NDEF", NDEFHandler);
private void NDEFHandler(ProximityDevice sender, ProximityMessage message)
{
     var NDefMessage = message.Data;
}

//write tag
publishId = device.PublishBinaryMessage("NDEF:WriteTag",
ndefMessage.ToByteArray().AsBuffer(), publishHandler);
```

Use it only for an advanced use. For example How to Create Cross-Platform LaunchApp NFC Tags

Note: when your uses a muliple-part NDEF message, you must use this protocol to get all parts. Other protocol consume only the first part.

Sample codes

Application Samples

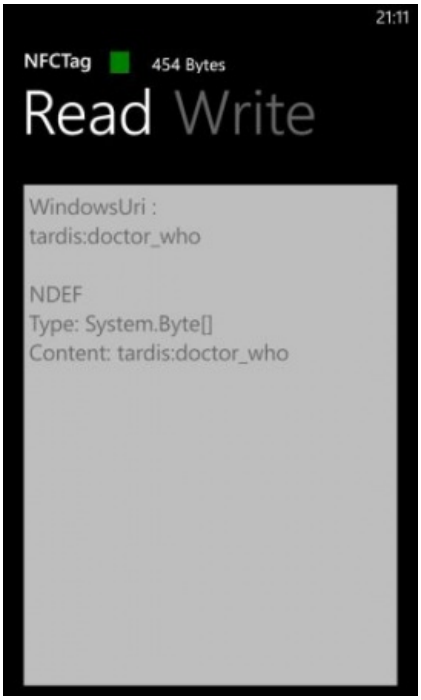
We have two sample codes focusing on two different aspects of NFC:

- Application that allows to demonstrate how to communicate between two phones:File:NFC WP8 sample app.zip
- Application that demonstrates use of each protocol to read and write NFC tag : Media:NFCTag.zip

This second application gives a code example for each protocol to read and write NFC tag. In order to build it you must install with NuGet NDEF Library for Proximity APIs (NFC) and Windows Phone Toolkit.

Note: This example uses extended functions provided by System.Runtime.InteropServices.WindowsRuntime namespace

(C) Copyright Nokia 2014. All rights reserved.



Write custom data or text

To write a custom data, you need to select the `Windows:WriteTag` protocol but be careful, a subtype is mandatory, we will use "NokiaTest" as subtype in the following example:

```
var dataWriter = new DataWriter() {UnicodeEncoding = UnicodeEncoding.Utf8};
dataWriter.WriteString("This is a test");

ProximityDevice.GetDefault().PublishBinaryMessage("Windows:WriteTag.NokiaTest",
dataWriter.DetachBuffer(), MesssageTransmitted);

private void MesssageTransmitted(ProximityDevice sender, long messageId)
{
  Deployment.Current.Dispatcher.BeginInvoke(() =>
  {
     MessageBox.Show("The message is written");
  });
}
```

Text data must be encoded in **UTF-8**.

To read this message, you need to subscribe to Windows.**<subtype>** messages (Windows.NokiaTest for the sample).

Write an url on the tag

It's pretty easy to write a link in an nfc tag, for example to share the address of your blog or your favorite website.

To do this, we will use the type `WindowsUri:WriteTag`, no subtype is necessary.

Uri must be encoded in **UTF16-LE**.

```
var dataWriter = new DataWriter() { UnicodeEncoding = UnicodeEncoding.Utf16LE };
dataWriter.WriteString("http://www.nokia.com");
ProximityDevice.GetDefault().PublishBinaryMessage("WindowsUri:WriteTag",
dataWriter.DetachBuffer(),MesssageTransmitted);

private void MesssageTransmitted(ProximityDevice sender, long messageId)
{
   Deployment.Current.Dispatcher.BeginInvoke(() =>
```


(C) Copyright Nokia 2014. All rights reserved.

```
    {
       MessageBox.Show("uri is written");
    });
}
```

When a windows phone will read this tag, Internet Explorer will be automatically launched.

Launch an application using tag

It's possible to write a tag that can be used to launch a specific app with optional launch parameters. The message must be a **UTF-16LE** encoded string where the values are delimited by tab characters or null values in the following form:

```
<launch arguments>[tab]<app platform 1>[tab]<app name 1>...[tab]<app platform
N>[tab]<app name N>
```

You must specify at least one app platform and app name.

For Windows Phone 8:

- The app platform is "WindowsPhone"
- The app name is the application id

To retrieve the app id of your application, you can find it on your Windows Phone Dev Center or on the www.windowsphone.com website. For example: http://www.windowsphone.com/en-us/store/app/wikipedia/fd40c569-2681-48df-9ff3-53ffd80aa9b5 the app id is {fd40c569-2681-48df-9ff3-53ffd80aa9b5}.

An example of this message is "param=test&toto=value\tWindowsPhone\{91000c5c-9943-43b8-aa65-7609d91057ef}". Of course, you can also support other app platforms.

```
var args = "param=test";
var appId="{91000c5c-9943-43b8-aa65-7609d91057ef}";
var launchAppMessage = args+ "\tWindowsPhone\t" + appId;

var dataWriter = new DataWriter() { UnicodeEncoding = UnicodeEncoding.Utf16LE };
dataWriter.WriteString(launchAppMessage);
ProximityDevice.GetDefault().PublishBinaryMessage("LaunchApp:WriteTag",
dataWriter.DetachBuffer(),MesssageTransmitted);

private void MesssageTransmitted(ProximityDevice sender, long messageId)
{
   Deployment.Current.Dispatcher.BeginInvoke(() =>
   {
       MessageBox.Show("The message is written");
   });
   ProximityDevice.GetDefault().StopPublishingMessage(messageId);
}
```

Retrieve parameters in your app

To retrieve the parameters of your app, you can write:

```
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            base.OnNavigatedTo(e);
            String value;
            NavigationContext.QueryString.TryGetValue("ms_nfp_launchargs", out value);
```


(C) Copyright Nokia 2014. All rights reserved.

```
        }
```

Save a contact on the tag

To describe a contact, you can use the **vcard** format. **vCard** is a file format standard for electronic business cards. They can contain name and address information, phone numbers, e-mail addresses, URLs, logos, photographs, and audio clips.

vCard specifications can be found here: http://tools.ietf.org/html/rfc6350

To write the vcard to the tag, we will use the The "WindowsMime:WriteTag" protocol with the subtype "text/vcard". Be careful the message must be a **UTF-16LE**.

```
var vcard = "BEGIN:VCARD\nVERSION:2.1\nFN:Jean Dupont\nN:Dupont;Jean\nTEL;CELL:+1234
56789\nEMAIL;INTERNET:jean.dupont@example.com\nEND:VCARD";
var dataWriter = new DataWriter() { UnicodeEncoding = UnicodeEncoding.Utf8 };
dataWriter.WriteString(vcard);
ProximityDevice.GetDefault().PublishBinaryMessage("WindowsMime:WriteTag.text/vcard",
dataWriter.DetachBuffer(), MesssageTransmitted);

private void MesssageTransmitted(ProximityDevice sender, long messageId)
{
   Deployment.Current.Dispatcher.BeginInvoke(() =>
   {
      MessageBox.Show("The message is written");
   });
}
```

Contributions

Useful content was merged in from the following articles (now redirected here):

- "Communication between two phones using NFC" by rudyhuyn
- "Write data to NFC tags using Windows Phone 8" by rudyhuyn

Interesting links

- Category:Near Field Communication (NFC)
- Proximity and NFC (Lumia Developer's Library)
  - Using NFC to exchange information
- Understanding NFC Data Exchange Format (NDEF) messages
- NFCInteractor FAQ
- NFC development
- NFC app scenarios
- How To Create a Facebook Like Button With A QR Code, Microsoft Tag or NFC
- Nfc Interactor + NDEF Library for Windows Phone 8
- How to Store Application Data on NFC Tags
- How to launch apps available in the store using NFC LaunchApp tag
- **Use NFC tags with Windows Phone 8**
- NFC on the Windows 8 Platform

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.