How to Launch Apps via Proximity APIs (NFC)

This article explains different ways of launching Windows Phone 8 apps using NFC and Proximity APIs.

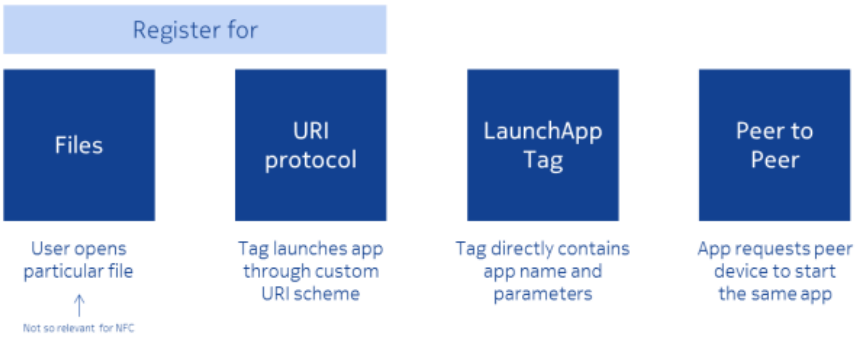
Introduction

Launching your own app via NFC is one of the most important use cases for developers – after all, you want your users to discover your app, have it on the screen whenever it could be relevant for the user, as well as let users easily share your app to other friends and / or let them create a seamless multi-user experience.

This article gives an overview of the possibilities as well as snippets how to get started with developing proximity apps. It is based on the slides of the Windows 8 Platform NFC Development lecture, but adds all the needed background information.

Launching Apps

Essentially, you have four main possibilities to launch your app on the Windows 8 Platform with NFC:



Registering for a specific file type to launch your app is also possible with NFC, but most likely not as relevant as the other possibilities. After all, NFC tags usually don't have enough writable space to store a whole file – they usually contain very efficient and standardized NDEF messages (see the article about acquiring content via NFC). Of course, your app can share a specific file type with another user via Proximity; but in most cases, you will use other means if you want to launch your app via NFC.

Custom URI Protocol

Registering for a custom URI scheme is one of the most powerful methods to launch your app in a wide variety of situations.

What is a custom URI scheme? Common examples are links on the web to make a call on Skype (*skype:mopius?call*) or to trigger a search on Spotify (*spotify:search:17th%20boulevard*). Also see the article about how to store geo coordinates on an NFC tag, which also contains examples on how to launch the maps application through a custom URL scheme.

Essentially, a custom URI scheme needs to contain your protocol / scheme name, followed by a colon character (":"). The remainder of the URI is the scheme-specific part, whose format is mostly up to your specific app. See Wikipedia for more details, for example with regards to constraints like reserved characters.

An example for a custom URI for your own app could be *nfcinteractor:compose*, which would launch your app and could then directly switch to the compose page, bypassing the standard start page. By adding more data to the scheme-specific part, you could also prefill the forms on the compose page, for example to be specific to the place where the URL was found.

That leaves the question: how can your app be launched via a protocol? If you register your app, the registration is system-wide. So it is even valid for a link on the web, discovered via the browser.

Relevant for NFC: URI messages received from a peer device via Proximity APIs (see this article for details on sending and receiving Uri messages), as well as URIs stored on NFC tags and discovered by tapping the tag.

In all those scenarios, your app would launch as soon as the device encounters the URI, or in case the app has not yet been installed, it will ask the user to visit the marketplace to find an appropriate app. In case this is a custom URI, this should in most cases be your app only (unless someone else chooses to

(C) Copyright Nokia 2013. All rights reserved.

use the same URI scheme).

MSDN contains an article that explains how to register for custom URI protocols in the Windows 8 platform.

LaunchApp Tags

In case you don't need a custom protocol and just want to launch your app when the user touches an NFC tag, there is a more direct way: creating a tag of the LaunchApp type. This also has the advantage that you can uniquely link to your app ID, without the potential danger of someone else putting an app into the store that implements the same protocol.

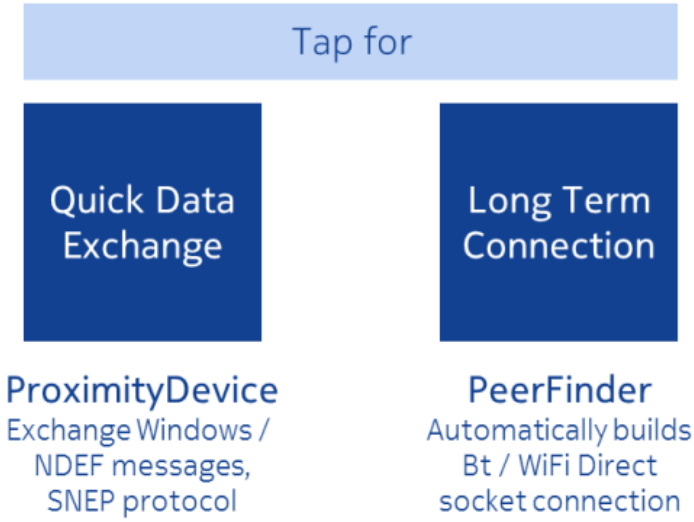
Unfortunately, this important use case hasn't yet been standardized by the NFC Forum. Therefore, Microsoft has defined its own extensible format, which can launch an app on any number of platforms. The format is quite easy, as seen on this diagram:

First, you specify the launch parameters – they are similar for all platforms, and are only specified once (tag space is very limited and therefore precious). This could for example be "view=compose". Afterwards, you specify one or more platforms names and the respective ID of your app on that platform. The Proximity APIs expect the input to be separated by tab characters. MSDN contains a short code sample that shows how to write such a LaunchApp tag for Windows 8 apps.

The resulting message that is written to the tag is an NDEF record of type absolute-URI, and the type name format is "windows.com/LaunchApp". The Proximity APIs will automatically reformat the tab-separated string that you pass to them, e.g., adding the string length in front of all the platform names, app ids, etc. You don't need to worry about those details, though; they all happen in the background.

Peer to Peer

For peer-to-peer scenarios, you have two different options to launch your app on the second device.



In case you only need a quick data exchange, you'd typically publish a message to the other phone, which should ideally use a custom URI scheme to also launch your app in case it's not running yet. In case you know that the other device also has your app running, you can send and listen to pretty much any kind of message (except a few reserved protocols, which can always be handled by the operating system). See this article for details on how to publish and subscribe messages between two devices with NFC.

The other scenario is to establish a longer-term multi-user session. In this case, NFC can only be used for establishing a permanent other communication channel. NFC itself is a very short-range technology, which works (depending on the antenna size) only up to 10 cm; in most real-life use cases, only up to 1 cm, given very small antenna sizes on mobile devices. Therefore, you'd actually tap the devices to exchange data. As soon as the users remove the devices again, the NFC link will be broken, and no data can be sent between the devices anymore.

The PeerFinder can automatically establish communication between two devices, and triggering the connection works either by tapping (via NFC) or by browsing for other peers. In case the other peer doesn't have the app opened / installed yet, the system will automatically offer the possibility to launch or install it. More details about this use case will be published in a separate article.

References

This article is based on the corresponding presentation by Andreas Jakl and provides background explanations that you wouldn't get when only browsing

(C) Copyright Nokia 2013. All rights reserved.

through the slides. The slides can be viewed and downloaded here: http://slidesha.re/UJAefK

To find out more about Proximity APIs, check out the API documentation.

For some cases also helpful is the specification for Proximity driver implementers – you should not need to know it, but it might help with some background knowledge if you want to go to lower levels or need details about encodings or limitations.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.