Advanced NFC

In this document

This document describes advanced NFC topics, such as working with various tag technologies, writing to NFC tags, and foreground dispatching, which allows an application in the foreground to handle intents even when other applications filter for the same ones.

Working with Supported Tag Technologies

When working with NFC tags and Android-powered devices, the main format you use to read and write data on tags is NDEF. When a device scans a tag with NDEF data, Android provides support in parsing the message and delivering it in an `NdefMessage` when possible. There are cases, however, when you scan a tag that does not contain NDEF data or when the NDEF data could not be mapped to a MIME type or URI. In these cases, you need to open communication directly with the tag and read and write to it with your own protocol (in raw bytes). Android provides generic support for these use cases with the `android.nfc.tech` package, which is described in Table 1. You can use the `getTechList()` method to determine the technologies supported by the tag and create the corresponding `TagTechnology` object with one of classes provided by `android.nfc.tech`

Table 1. Supported tag technologies

Class	Description
`TagTechnology`	The interface that all tag technology classes must implement.
`NfcA`	Provides access to NFC-A (ISO 14443-3A) properties and I/O operations.
`NfcB`	Provides access to NFC-B (ISO 14443-3B) properties and I/O operations.
`NfcF`	Provides access to NFC-F (JIS 6319-4) properties and I/O operations.
`NfcV`	Provides access to NFC-V (ISO 15693) properties and I/O operations.
`IsoDep`	Provides access to ISO-DEP (ISO 14443-4) properties and I/O operations.
`Ndef`	Provides access to NDEF data and operations on NFC tags that have been formatted as NDEF.
`NdefFormatable`	Provides a format operations for tags that may be NDEF formattable.

The following tag technlogies are not required to be supported by Android-powered devices.

Table 2. Optional supported tag technologies

Class	Description
`MifareClassic`	Provides access to MIFARE Classic properties and I/O operations, if this Android device supports MIFARE.
`MifareUltralight`	Provides access to MIFARE Ultralight properties and I/O operations, if this Android device supports MIFARE.

Working with tag technologies and the ACTION_TECH_DISCOVERED intent

When a device scans a tag that has NDEF data on it, but could not be mapped to a MIME or URI, the tag dispatch system tries to start an activity with the `ACTION_TECH_DISCOVERED` intent. The `ACTION_TECH_DISCOVERED` is also used when a tag with non-NDEF data is scanned. Having this fallback allows you to work with the data on the tag directly if the tag dispatch system could not parse it for you. The basic steps when working with tag technologies are as follows:

1. Filter for an `ACTION_TECH_DISCOVERED` intent specifying the tag technologies that you want to handle. See Filtering for NFC intents for more information. In general, the tag dispatch system tries to start a `ACTION_TECH_DISCOVERED` intent when an NDEF message cannot be mapped to a MIME type or URI, or if the tag scanned did not contain NDEF data. For more information on how this is determined, see The Tag Dispatch System.
2. When your application receives the intent, obtain the `Tag` object from the intent:

   ```
   Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
   ```

3. Obtain an instance of a `TagTechnology`, by calling one of the `get` factory methods of the classes in the `android.nfc.tech` package. You can enumerate the supported technologies of the tag by calling `getTechList()` before calling a `get` factory method. For example, to obtain an instance of `MifareUltralight` from a `Tag`, do the following:

   ```
   MifareUltralight.get(intent.getParcelableExtra(NfcAdapter.EXTRA_TAG));
   ```

Reading and writing to tags

Reading and writing to an NFC tag involves obtaining the tag from the intent and opening communication with the tag. You must define your own protocol stack to read and write data to the tag. Keep in mind, however, that you can still read and write NDEF data when working directly with a tag. It is up to you how you want to structure things. The following example shows how to work with a MIFARE Ultralight tag.

```
package com.example.android.nfc;

import android.nfc.Tag;
import android.nfc.tech.MifareUltralight;
import android.util.Log;
import java.io.IOException;
import java.nio.charset.Charset;

public class MifareUltralightTagTester {

    private static final String TAG = MifareUltralightTagTester.class.getSimpleName();

    public void writeTag(Tag tag, String tagText) {
        MifareUltralight ultralight = MifareUltralight.get(tag);
        try {
            ultralight.connect();
            ultralight.writePage(4, "abcd".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(5, "efgh".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(6, "ijkl".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(7, "mnop".getBytes(Charset.forName("US-ASCII")));
        } catch (IOException e) {
            Log.e(TAG, "IOException while closing MifareUltralight...", e);
        } finally {
            try {
                ultralight.close();
            } catch (IOException e) {
                Log.e(TAG, "IOException while closing MifareUltralight...", e);
            }
        }
    }

    public String readTag(Tag tag) {
        MifareUltralight mifare = MifareUltralight.get(tag);
        try {
            mifare.connect();
            byte[] payload = mifare.readPages(4);
            return new String(payload, Charset.forName("US-ASCII"));
        } catch (IOException e) {
            Log.e(TAG, "IOException while writing MifareUltralight
            message...", e);
        } finally {
            if (mifare != null) {
                try {
                    mifare.close();
                }
                catch (IOException e) {
                    Log.e(TAG, "Error closing tag...", e);
                }
            }
        }
        return null;
    }
}
```

Using the Foreground Dispatch System

The foreground dispatch system allows an activity to intercept an intent and claim priority over other activities that handle the same intent. Using this system involves constructing a few data structures for the Android system to be able to send the appropriate intents to your application. To enable the foreground dispatch system:

1. Add the following code in the `onCreate()` method of your activity:
    1. Create a `PendingIntent` object so the Android system can populate it with the details of the tag when it is scanned.

        ```
        PendingIntent pendingIntent = PendingIntent.getActivity(
            this, 0, new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        ```

    2. Declare intent filters to handle the intents that you want to intercept. The foreground dispatch system checks the specified intent filters

with the intent that is received when the device scans a tag. If it matches, then your application handles the intent. If it does not match, the foreground dispatch system falls back to the intent dispatch system. Specifying a `null` array of intent filters and technology filters, specifies that you want to filter for all tags that fallback to the `TAG_DISCOVERED` intent. The code snippet below handles all MIME types for `NDEF_DISCOVERED`. You should only handle the ones that you need.

```
IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
    try {
        ndef.addDataType("*/*");    /* Handles all MIME based dispatches.
                                       You should specify only the ones that you need. */
    }
    catch (MalformedMimeTypeException e) {
        throw new RuntimeException("fail", e);
    }
   intentFiltersArray = new IntentFilter[] {ndef, };
```

3. Set up an array of tag technologies that your application wants to handle. Call the `Object.class.getName()` method to obtain the class of the technology that you want to support.

```
techListsArray = new String[][] { new String[] { NfcF.class.getName() } };
```

2. Override the following activity lifecycle callbacks and add logic to enable and disable the foreground dispatch when the activity loses (`onPause()`) and regains (`onResume()`) focus. `enableForegroundDispatch()` must be called from the main thread and only when the activity is in the foreground (calling in `onResume()` guarantees this). You also need to implement the `onNewIntent` callback to process the data from the scanned NFC tag.

```
public void onPause() {
    super.onPause();
    mAdapter.disableForegroundDispatch(this);
}

public void onResume() {
    super.onResume();
    mAdapter.enableForegroundDispatch(this, pendingIntent, intentFiltersArray, techListsArray);
}

public void onNewIntent(Intent intent) {
    Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
    //do something with tagFromIntent
}
```

See the ForegroundDispatch sample from API Demos for the complete sample.