

NFC Integration Guide

Overview

NFC (Near Field Communication) allows users to quickly launch the app by tapping an NFC tag, which automatically parses the Bluetooth MAC address stored in the tag and directly connects to the printer. The overall workflow is as follows:

```
User taps NFC tag
→ App is launched (cold start / warm start)
→ Parse the BMAC parameter in the URL
→ SDK Bluetooth connects to the printer
→ UI updates connection status
```

Example URL format stored in the NFC tag:

```
http://hprint.hpert.com/hereset/index.html?Model=RTP-
808BW&SN=R808BW262000442&BMAC=28:D4:1E:73:A4:19
```

The `BMAC` parameter is the printer's Bluetooth MAC address.

Step 1: Configure AndroidManifest.xml

1.1 Add NFC Permission

Add the following inside the `<manifest>` tag:

```
<uses-permission android:name="android.permission.NFC" />
```

1.2 Declare NFC Hardware Feature (Optional)

Add `required="false"` to allow the app to run on devices without NFC hardware:

```
<uses-feature android:name="android.hardware.nfc" android:required="false" />
```

1.3 Set Activity Launch Mode

To ensure only one instance of the Activity exists when launched via NFC and avoid duplicate creation, set `launchMode="singleTask"`:

```

<activity
    android:name=".MainActivity"
    android:launchMode="singleTask"
    android:exported="true"
    android:label="@string/app_name">

```

1.4 Add NFC NDEF_DISCOVERED Intent Filter

This intent-filter matches NDEF URI records in NFC tags. When a tag contains a URL in the format `http://hprint.hpert.com/...`, the system dispatches the Intent to this Activity:

```

<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
    <data android:scheme="https" />
    <data android:host="hprint.hpert.com" />
</intent-filter>

```

Complete AndroidManifest Reference

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- NFC Permission -->
    <uses-permission android:name="android.permission.NFC" />
    <uses-feature android:name="android.hardware.nfc" android:required="false" />

    <!-- Keep existing Bluetooth, USB permissions unchanged -->
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application ...>
        <activity
            android:name=".MainActivity"
            android:launchMode="singleTask"
            android:exported="true"
            android:label="@string/app_name">

            <!-- Home screen launcher entry -->
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

```

```

<!-- NFC NDEF_DISCOVERED -->
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
    <data android:scheme="https" />
    <data android:host="hprint.hpdt.com" />
</intent-filter>

<!-- Deep Link -->
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="http"
          android:host="hprint.hpdt.com"
          android:pathPrefix="/hereset/index.html" />
</intent-filter>
</activity>
</application>
</manifest>

```

Step 2: Handle NFC Intent

There are two scenarios when NFC tapping launches the app:

- **Cold start:** The app is not running. The system creates a new process and calls `onCreate()`, where the Intent is passed via the `intent` parameter.
- **Warm start:** The app is already running in the background. The system calls `onNewIntent()` with the new Intent.

Therefore, NFC Intent handling is required in both entry points.

2.1 Cold Start Handling (onCreate)

Call the NFC handling method in `onCreate`:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // ... other initialization code ...

    // Handle NFC Intent on cold start
    handleNfcIntent(intent)
}

```

2.2 Warm Start Handling (onNewIntent)

Override the `onNewIntent` method:

```
override fun onNewIntent(intent: Intent) {
    super.onNewIntent(intent)
    handleNfcIntent(intent)
}
```

2.3 Core NFC Intent Handling Method

```
private fun handleNfcIntent(intent: Intent) {
    // Only handle NDEF_DISCOVERED and VIEW actions
    val action = intent.action
    if (action != NfcAdapter.ACTION_NDEF_DISCOVERED && action !=
Intent.ACTION_VIEW) {
        return
    }

    val uri = intent.data ?: return
    val url = uri.toString()
    Log.d("TAG", "NFC/DeepLink received: $url")

    // Extract Bluetooth MAC address from URL
    val mac = parseMacFromNfcUrl(url)
    if (mac == null) {
        Toast.makeText(this, "Invalid NFC data, printer address not found",
Toast.LENGTH_LONG).show()
        return
    }

    Toast.makeText(this, "NFC discovered printer: $mac",
Toast.LENGTH_SHORT).show()

    // Start Bluetooth connection
    isConnectingState.value = true
    loadingMessageState.value = "NFC Bluetooth connecting..."
    lifecycleScope.launch {
        try {
            connectBluetooth(mac)
            updateConnectionStatus()
        } finally {
            isConnectingState.value = false
        }
    }
}
```

Step 3: Parse Bluetooth MAC Address from URL

The URL in the NFC tag contains the printer's Bluetooth MAC address, passed via the `BMAC` query parameter.

URL format:

```
http://hprint.hpdt.com/hereset/index.html?Model=RTP-808BW&SN=R808BW262000442&BMAC=28:D4:1E:73:A4:19
```

Parsing method:

```
/**
 * Parse BMAC (Bluetooth MAC address) from NFC URL
 * URL format: http://hprint.hpdt.com/hereset/index.html?Model=xxx&SN=xxx&BMAC=xx:xx:xx:xx:xx:xx
 */
private fun parseMacFromNfcUrl(url: String): String? {
    return try {
        val uri = Uri.parse(url)
        uri.getQueryParameter("BMAC")
    } catch (e: Exception) {
        Log.e("TAG", "Failed to parse NFC URL", e)
        null
    }
}
```

Use `Uri.getQueryParameter("BMAC")` to directly extract the specified parameter value — no manual regex matching required.

Step 4: Bluetooth Connection via SDK

After parsing the MAC address, call the SDK's Bluetooth connection interface to connect.

4.1 Bluetooth Connection Method

```
private suspend fun connectBluetooth(address: String) {
    // Set connecting state
    ConnectionHolder.setConnecting()

    // Create Bluetooth connection object
    val connection = BluetoothConnection()

    // Execute connection
    when (val result =
connection.connect(ConnectionParams.Bluetooth(address))) {
        is Result.Success -> {
            // Connection successful, save connection state
            ConnectionHolder.setConnection(
```

```

        connection,
        ConnectionHolder.ConnectionType.BLUETOOTH,
        address
    )
    Toast.makeText(this, "Connected successfully",
    Toast.LENGTH_SHORT).show()
}
is Result.Failure -> {
    // Connection failed, save error info
    ConnectionHolder.setError(result.error.message)
    Toast.makeText(
        this,
        "Connection failed: ${result.error.message}",
        Toast.LENGTH_SHORT
    ).show()
}
}
}
}

```

4.2 ConnectionHolder Overview

`ConnectionHolder` is a global singleton that manages printer connection state, providing the following capabilities:

Method	Description
<code>setConnection(connection, type, info)</code>	Set connected state
<code>setConnecting()</code>	Set connecting state
<code>setError(message)</code>	Set connection failed state
<code>isConnected()</code>	Check if connected
<code>disconnect()</code>	Disconnect

Connection type enum:

```

enum class ConnectionType {
    NONE, BLUETOOTH, WIFI, USB, SERIAL
}

```

Step 5: UI Updates

During and after the connection process, update the UI to provide feedback to the user.

5.1 Connecting State

Use a loading dialog to indicate connection in progress:

```
isConnectingState.value = true
loadingMessageState.value = "NFC Bluetooth connecting..."
```

Display the loading dialog in Compose UI:

```
if (isLoading) {
    AlertDialog(
        onDismissRequest = {},
        confirmButton = {},
        title = { Text("Please wait...") },
        text = {
            Row(verticalAlignment = Alignment.CenterVertically) {
                CircularProgressIndicator(modifier = Modifier.size(20.dp))
                Spacer(modifier = Modifier.width(8.dp))
                Text(loadingMessage)
            }
        }
    )
}
```

5.2 Connection Completed State

Call `updateConnectionStatus()` to refresh the UI after connection is complete:

```
private fun updateConnectionStatus() {
    isConnectedState.value = ConnectionHolder.isConnected()
    connectionStatusText.value = if (isConnectedState.value) {
        "Connected: ${ConnectionHolder.connectionType.name}"
    } else {
        "Please connect printer"
    }
}
```

The status card in the UI automatically changes color based on connection state (connected = theme color, disconnected = gray).

5.3 Error Prompt

Display specific error information via Toast on connection failure to help users troubleshoot.

Complete Code Reference

Full NFC-related code in MainActivity:

```

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ... initialization code ...

        // Handle NFC Intent on cold start
        handleNfcIntent(intent)
    }

    override fun onNewIntent(intent: Intent) {
        super.onNewIntent(intent)
        handleNfcIntent(intent)
    }

    /**
     * Handle NFC Intent: parse URL, extract MAC, Bluetooth connection
     */
    private fun handleNfcIntent(intent: Intent) {
        val action = intent.action
        if (action != NfcAdapter.ACTION_NDEF_DISCOVERED && action !=
Intent.ACTION_VIEW) {
            return
        }

        val uri = intent.data ?: return
        val url = uri.toString()
        Log.d("TAG", "NFC/DeepLink received: $url")

        val mac = parseMacFromNfcUrl(url)
        if (mac == null) {
            Toast.makeText(this, "Invalid NFC data, printer address not
found", Toast.LENGTH_LONG).show()
            return
        }

        Toast.makeText(this, "NFC discovered printer: $mac",
Toast.LENGTH_SHORT).show()

        // Start Bluetooth connection
        isConnectingState.value = true
        loadingMessageState.value = "NFC Bluetooth connecting..."
        lifecycleScope.launch {
            try {
                connectBluetooth(mac)
                updateConnectionStatus()
            } finally {
                isConnectingState.value = false
            }
        }
    }
}

```

```

    }
}

/**
 * Parse BMAC from NFC URL
 * URL: http://hprint.hpert.com/hereset/index.html?
Model=xxx&SN=xxx&BMAC=xx:xx:xx:xx:xx:xx
 */
private fun parseMacFromNfcUrl(url: String): String? {
    return try {
        val uri = Uri.parse(url)
        uri.getQueryParameter("BMAC")
    } catch (e: Exception) {
        Log.e("TAG", "Failed to parse NFC URL", e)
        null
    }
}

/**
 * SDK Bluetooth connection
 */
private suspend fun connectBluetooth(address: String) {
    ConnectionHolder.setConnecting()
    val connection = BluetoothConnection()
    when (val result =
connection.connect(ConnectionParams.Bluetooth(address))) {
        is Result.Success -> {
            ConnectionHolder.setConnection(
                connection,
                ConnectionHolder.ConnectionType.BLUETOOTH,
                address
            )
            Toast.makeText(this, "Connected successfully",
Toast.LENGTH_SHORT).show()
        }
        is Result.Failure -> {
            ConnectionHolder.setError(result.error.message)
            Toast.makeText(
                this,
                "Connection failed: ${result.error.message}",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}
}
}
}

```

Required Imports

```
import android.nfc.NfcAdapter
import android.net.Uri
import android.bluetooth.BluetoothDevice
import com.printer.sdk.connection.BluetoothConnection
import com.printer.sdk.connection.ConnectionParams
import com.printer.sdk.model.Result
```

Important Notes

1. **Android 12+ (API 31+) Bluetooth Permissions:** If `targetSdk >= 31`, both `BLUETOOTH_CONNECT` and `BLUETOOTH_SCAN` permissions must be declared and requested at runtime.
2. **Location Permission:** Bluetooth scanning requires `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION` permission (Android 11 and below).
3. **NFC Permission is a Normal Permission:** The `NFC` permission is at the `normal` level — simply declare it in `AndroidManifest`, no runtime request required.
4. **Emulator Testing:** Android emulators do not support NFC by default. It is recommended to test on a physical device.
5. **URL Matching Scope:** In the `NDEF_DISCOVERED` intent-filter, using only `scheme=http` and `host=hprint.hpnt.com` without `pathPrefix` will match all paths under that domain, making it more versatile.
6. **Connection Callback:** Bluetooth connection is a time-consuming operation. Always execute it within a coroutine to avoid blocking the UI thread.