distriqt

// tutorial

Using the Android Native Maps Extension

Version 1.0 by Shane Korin

This tutorial will cover setting up and running the Android version of the NativeMaps ANE from distriqt.

The NativeMaps ANE utilises the Google Maps Android API v2. API keys for version 1 of Android Google Maps will not work with this ANE.

This ANE requires a minimum of Android API 12 (Android 3.2) or later. Earlier versions of Android are not supported at this stage.

Due to the way Flash Builder signs APK files, it's currently quite difficult to debug an application that uses the native maps directly from Flash Builder or the Flash IDE. As such, you will be required to build and package your application using ADT on the command line in order to debug or release your application.

We have provided this guide along with command line tools to help with this process.

As of Flash Builder 4.7, there are options available in the IDE to help with setting these correct signing options, however this doesn't yet appear to work reliably so this guide covers using the command line instead.

We'll be doing our best to build tooling and/or additional support to enable IDE building and debugging after the current release.

1. Understanding certificates

The Google Maps API requires integration with the Android certificates on your machine. There are two types of certificates - debug and release.

The release certificate for AIR for Android applications is a P12 file. For simplicity, we'll use the same certificate for signing and testing the application in debug mode. (A guide for using the default android debug key can be found later in this guide, if desired).

****
UPDATE
We are now providing instructions to allow you to run your Android maps application in debug mode directly from Flash Builder. Previously we had issues getting this to work, however we're now confident this is possible. See section 5.1 below in this document for instructions.
****

If you do not already have a P12 certificate created to use for your application, you should create one now.

A certificate can be created using the ADT command in Terminal or on the command line in Windows. Adobe has a guide for this command here:
http://help.adobe.com/en_US/air/build/WS901d38e593cd1bac1e63e3d128fc240122-7ffc.html

NOTE: *Make sure to set a minimum value of 25 for the "validityPeriod" when creating the certificate.*

You will need to specify the SHA1 fingerprint of your certificate along with your application ID for the Google Maps API to allow your app to access the maps service.

To obtain this fingerprint, open a command prompt and navigate to the path of your P12 certificate file, and run the following command. Note that you must change the values in red to match the appropriate values for your certificate.

```
keytool -list -keystore myCertificate.p12 -storepass password -storetype PKCS12 -v
```

This command should print out information about your debug certificate. Take note of the SHA1 fingerprint value, as this will be required later.

NOTE*: On Windows, if they keytool is not in your PATH environment variable, the tool should be located in the following location: "C:\Program Files\Java\jre7\bin" or similar. If not, you may need to install Java.*

2. Register an API key

You must register an API key for your application. In a browser, go to
https://code.google.com/apis/console/.

You should create a new API project here for your Maps application. If you have no other projects, you will be immediately prompted to create one. If you have existing applications, click the project name dropdown in the top left and choose "Create". Give your project a name and click "Create Project".

Next you'll need to enable the Maps API for your project. If you were not automatically directed to the Services page, click the "Services" menu item. Scroll down to find the item named "Google Maps Android API v2”. Turn the switch to “ON”.

Now click the “API Access” menu item. You will see you already have an API key for browser apps. You must also create a key for your Android app. Click the “Create new Android key...” button at the bottom. This will open a popup box where you can enter the SHA1 fingerprint and application ID for your application.

Here you need to enter the SHA1 fingerprint from step 1, followed by a semicolon (;), followed by your application ID.

An example of this value would look similar to:
`BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:90:AF:A1:66:6E:44:5D:75;air.com.distriqt.test.debug`

The application ID after the semicolon represents your application ID which you are using for your AIR application. One thing to note is that you must prefix your application ID with “air.”. For example, if your application ID is `com.something.myApp`, you would enter `air.com.something.myApp`.

Once you have entered this information, click “Create”. You will now see your Android API key listed under the Android app keys section. Make a note of your API key as you’ll need it soon.

3. Set up your application descriptor

Next you will need to make some changes to your app descriptor XML file to enable the Maps API to work and communicate with your app.

Note: You can refer to the example application provided with our NativeMaps ANE to see an example of the correct descriptor setup.

Open the application descriptor XML file for your application. For this example we are assuming that your application ID is `com.something.myApp`.

This is an example of the required <android> section from a descriptor file. Note that any values highlighted in red are values you must modify for your application.

```
<android>
<manifestAdditions><![CDATA[
<manifest android:installLocation="auto">
        <uses-sdk android:minSdkVersion="12" android:targetSdkVersion="17" />
        <uses-permission android:name="android.permission.INTERNET"/>
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
        <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>

        <!-- Replace the 'com.something.myApp’ id string here with your app package ID -->
        <!-- Make sure to include 'air.' at the beginning of the package name -->
        <permission
```

```
                android:name="air.com.something.myApp.permission.MAPS_RECEIVE"
                android:protectionLevel="signature"/>
        <uses-permission android:name="air.com.something.myApp.permission.MAPS_RECEIVE"/>

        <uses-feature android:glEsVersion="0x00020000" android:required="true"/>
        <application>
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>

                <activity android:name="com.distriqt.extension.nativemaps.NativeMapActivity"
                android:theme="@android:style/Theme.Translucent.NoTitleBar"></activity>

                <!-- Replace the API key with your Android Google Maps v2 api key -->
                <meta-data
                        android:name="com.google.android.maps.v2.API_KEY"
                        android:value="AIzaSyAQ56O2jVPuqkR_hDsg59W8H5PCLmIRqo4" />

                <uses-library android:name="com.google.android.maps"/>
        </application>
</manifest>
]]></manifestAdditions>
</android>
```

4. Creating a Map

To create a map in your application, first initialise the NativeMap ANE, and then call the NativeMaps.service.createMap function.

Short example code:

```
// Your developer key is the distriqt native extension key provided when you purchased the package.
NativeMaps.init( YOUR_DEVELOPER_KEY );

if (NativeMaps.isSupported)
{
        // Create a map 480px wide and 400px high in the top-left corner of the stage.
        NativeMaps.service.createMap( 480, 400, 0, 0 );
}
else
{
        // Not supported
        trace(“NativeMaps not supported on this platform.”);
}
```

5. Building your Application

****

UPDATE

We now provide instructions for debugging directly from Flash Builder. See section 5.1 below for this information. You can skip section 5 if you wish to debug directly from the IDE.

****

When you're ready to test the application, you must build the APK using ADT on the command line. (Terminal on OSX or command prompt on Windows).

The ADT command to compile your APK is as follows. Note the red items which must be modified for your application.

```
adt -package -target apk-debug -storetype pkcs12 -keystore myCertificate.p12 -storepass password
MyApp.apk MyApp-app.xml MyApp.swf icons/ res/ -extdir /path/to/ANE/folder/
```

If the ADT tool is not in your PATH environment variables, it can be located in a similar location to these example paths:

Flash Builder 4.7 (with default AIRSDK)

Mac

```
/Applications/Adobe Flash Builder 4.7/eclipse/plugins/com.adobe.flash.compiler_4.7.0.349722/AIRSDK/bin
```

Windows

```
C:\Program Files\Adobe\Adobe Flash Builder
4.7\eclipse\plugins\com.adobe.flash.compiler_4.7.0.349722\AIRSDK\bin
```

Flash Builder 4.6 or other

The ADT tool will be located in the SDK path you're using to compile, under the "/bin" directory.

We have provided scripts to help with this process (Windows versions are suffixed with 'Win'. These scripts are located in the example project folder, under the "tools" directory. You must copy the scripts you wish to use into the bin-debug or build folder for your application. Also, open them with a text editor, and modify the parameters to suit your application values and development environment.

build

This script just builds your APK in debug mode using ADT. Open the build file in a text editor, and you will see parameters defined at the top of the file which you should modify to match your application and environment settings.

buildAndRun

This script builds the application the same as above, but also attempts to install and run the APK on a connected device. You will need to modify the same parameters as above in this file, and also modify the path to the ADB tool in the three ADB commands at the bottom. If ADB is not added to your PATH, the ADB tool is part of the Android SDK and is contained in the /platform-tools directory of the Android SDK.

buildRelease
This script builds a release version of your application. Again, open with a text editor and modify the parameters at the top accordingly.

NOTE: *Depending on your PATH setup you may need to modify the path to ADT and ADB in the example scripts*

5.1. Debugging directly from Flash Builder

If desired, you can debug your maps application directly from Flash Builder without having to use the command line tools. In order to do this, you must simply get the SHA1 fingerprint of the default signing certificate used by Flash Builder, rather than your own keystore. (Read more about the keystore certificates in Section 1 of this document).

First, open a Terminal window and navigate to the path where Flash Builder stores its default p12 file. On Mac, this path will be something similar to:

```
/Applications/Adobe Flash Builder
4.7/eclipse/plugins/com.adobe.flexide.multiplatform.android_4.7.0.349722/resources
```

The version number of the "android_XXXX" folder may differ, you'll need to browse and find this.

The **resources** folder here should contain a file called **debug-certificate-android.p12**. Follow the same process as described in Section 1, with slightly different parameters:

```
keytool -list -keystore debug-certificate-android.p12 -storepass debug -storetype PKCS12 -v
```

Once you have the SHA1 fingerprint for the debug certificate here, follow the same steps as described in Section 2 to add the key to your Google API entries to allow an application access to the maps when it's signed with this default debug p12 file through Flash Builder.

6. Using the default Android Keystore for Signing

NOTE: *This is optional. The above steps for using the p12 certificate negate the need for this unless* *specifically required in your case.*

If desired, you can also use the default Android keystore for signing a debug version of your application.

The debug keystore file will usually be located at the following path:
OSX: `/Users/<username>/.android/debug.keystore`
Windows: `C:\Users\<username>\.android\debug.keystore`

Again you must get the SHA1 fingerprint from this certificate and enter this to the Google APIs Console for your project.

To display the SHA-1 fingerprint for your debug certificate, you can use the following command in a Terminal window, or on the command line in Windows:

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass
android
```

By default the storepass and keypass values should be "android" unless you have changed them.

Then to build your APK, a slightly different ADT command is required, as follows:

```
adt -package -target apk-debug -storetype jks -keystore ~/.android/debug.keystore -storepass android
MyApp.apk MyApp-app.xml MyApp.swf icons/ res/ -extdir /path/to/ANE/folder/
```

7. Common Problems

Grey Map / No Tiles appearing

The most common problem you may encounter is the map frame being displayed, but only grey tiles showing up with no visible map. This problem is caused by incorrect permissions settings in terms of your Google Maps API key. Double check the following things:
 - The Android manifest settings in your descriptor XML are correct, and your API key is entered correctly (see example app)
 - Your application ID and SHA1 certificate fingerprint are correct and set up correctly in the Google Maps API console under your project.