Using the Binary - Android

Versions

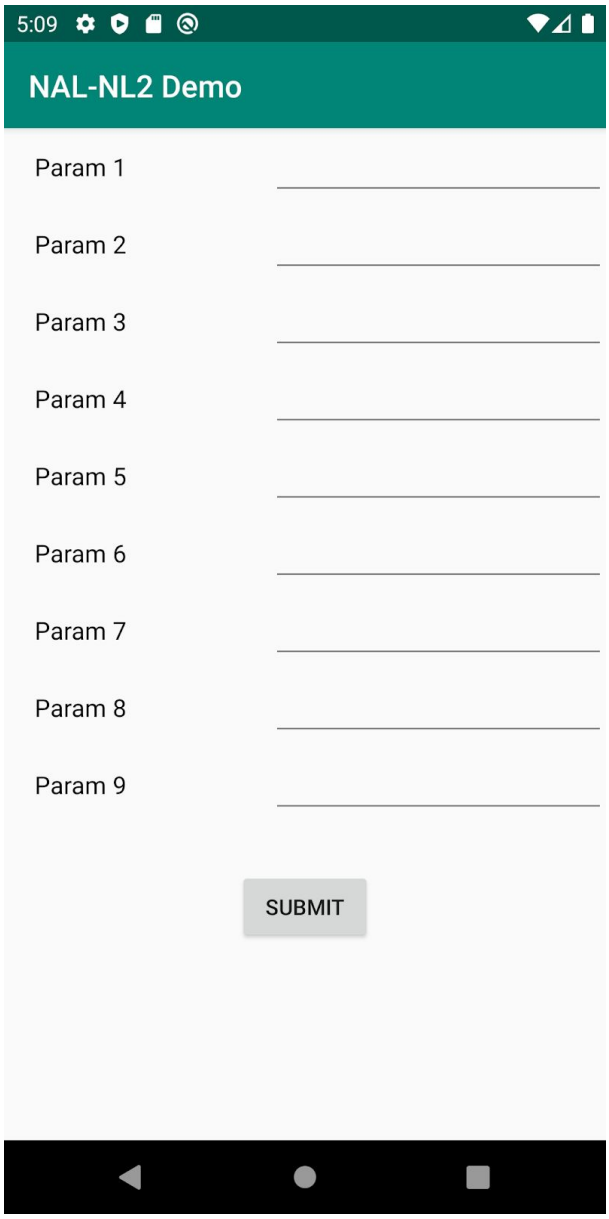
This Android project was built and compiled using Android Studio 4.0.1, using Android API 29.
API 29 (Android version 10) is chosen to ensure the maximum future capability.
This source code can be compiled against earlier versions of Android if required.

Building the APP

1. Open `Android NAL-NL2 Demo` in Android Studio
2. Build the project: `Build > Make Project` - using the menu
3. After a successful build, the compiled APK will reside in `./Android NAL-NL2 Demo/app/build/outputs/debug/app-debug.apk`

Running the APP

1. Open `Android NAL-NL2 Demo` in Android Studio
2. Run the project: `Run > Run "app"` – using the menu
3. When the project is successfully built & installed, the app will open in an Android emulator - Our team used a “Pixel 3 with API 29” for this test



Screenshot of the example APP on Android Pixel 3

How to reference the Binary

1. Place `nl2-release.aar` into `/app/libs` folder
2. Reference the binary file in `app/build.gradle`, by adding to the `android` section

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

3. Reference the binary file in `app/build.gradle`, by adding to the `dependencies` section

```
implementation (name: "nl2-release", ext:"aar")
```

4. Verify using the example provided in `Android NAL-NL2 Demo/app/build.gradle`

For the classes you would like to use the NAL binary, add this to the top of the file to import the required namespaces

```
import au.org.nal.NativeManager;
import au.org.nal.OutputResult;
```

Example of the namespace import: `Android NAL-NL2 Demo/app/src/main/java/au/org/nal/nal_nl2_demo/MainActivity.java`

To use the API

Once the binary is correctly referenced and its namespace imported, you can set the right parameters for invocation and then call the API.

```
NativeManager.getInstance(this).SetAdultChild(client.adultChild
, client.dateOfBirth);
NativeManager.getInstance(this).SetExperience(client.experience
);
NativeManager.getInstance(this).SetCompSpeed(compSpeed);
NativeManager.getInstance(this).SetTonalLanguage(client.tonal);
NativeManager.getInstance(this).SetGender(client.gender);
NativeManager.getInstance(this).CrossOverFrequencies_NL2(CFArra
y, channels, ac, ac, freqInCh);
NativeManager.getInstance(this).setBWC(channels, CFArray);
NativeManager.getInstance(this).CompressionThreshold_NL2(ct,
bandwidth, selection, WBCT, haType, direction, mic, calcCh);
```

```
OutputResult outputResult =
NativeManager.getInstance(this).RealEarAidedGain_NL2(data, ac,
ac, level, limiting, channels, direction, mic, ac, numAids);
```

You can obtain and examine the output from the API call by using:

```
double[] data2 = outputResult.getOutput1();

Log.i(TAG," 125Hz - " + data2[0]);
Log.i(TAG," 160Hz - " + data2[1]);
Log.i(TAG," 200Hz - " + data2[2]);
Log.i(TAG," 250Hz - " + data2[3]);
Log.i(TAG," 315Hz - " + data2[4]);
Log.i(TAG," 400Hz - " + data2[5]);
Log.i(TAG," 500Hz - " + data2[6]);
Log.i(TAG," 630Hz - " + data2[7]);
Log.i(TAG," 800Hz - " + data2[8]);
Log.i(TAG," 1000Hz - " + data2[9]);
Log.i(TAG," 1250Hz - " + data2[10]);
Log.i(TAG," 1600Hz - " + data2[11]);
Log.i(TAG," 2000Hz - " + data2[12]);
Log.i(TAG," 2500Hz - " + data2[13]);
Log.i(TAG," 3150Hz - " + data2[14]);
Log.i(TAG," 4000Hz - " + data2[15]);
Log.i(TAG," 5000Hz - " + data2[16]);
Log.i(TAG," 6300Hz - " + data2[17]);
Log.i(TAG," 8000Hz - " + data2[18]);
```

Due to the Android limitations, API output is mapped according to the table below:

API	API Name	API Outputs	Android Outputs
2	RealEarInsertionGain_NL2	REIG[] - double	REIG[] -> getOutput1()
3	RealEarAidedGain_NL2	REAG - double	REAG -> getOutput1()
4	TccCouplerGain_NL2	TccCG - double lineType[] - int	TccCG -> getOutput1() lineType[] -> getOutput2b()
5	EarSimulatorGain_NL2	ESG[] - double lineType - int	ESG[] -> getOutput1() lineType -> getOutput2b()

6	RealEarInputOutputCurve_NL2	REIO[] - double REIOunl[] - double	REIO[] -> getOutput1() REIOunl[] -> getOutput2()
7	TccInputOutputCurve_NL2	TccIO[] - double TccIOunl[] - double lineType - int	TccIO[] -> getOutput1() TccIOunl[] -> getOutput2() lineType -> getOutput3b()
8	EarSimulatorInputOutputCurve_NL2	ESIO[] - double ESIOunl[] - double lineType - int	ESIO[] -> getOutput1() ESIOunl[] -> getOutput2() lineType -> getOutput3b()
9	Speech_o_Gram_NL2	Speech_rms[] - double Speech_max[] - double Speech_min[] - double Speech_thresh[] - double	Speech_rms[] -> getOutput1() Speech_max[] -> getOutput2() Speech_min[] -> getOutput3() Speech_thresh[] -> getOutput4()
10	AidedThreshold_NL2	AT[] - double	AT[] -> getOutput1()
11	GetREDDindiv	REDD[] - double	REDD[] -> getOutput1()
12	GetREDDindiv9	REDD[] - double	REDD[] -> getOutput1()
13	GetREURindiv	REUR[] - double	REUR[] -> getOutput1()
14	GetREURindiv9	REUR[] - double	REUR[] -> getOutput1()
19	CrossOverFrequencies_NL2	CFArray[] - double FreqInCh[] - int	CFArray[] -> getOutput1() FreqInCh[] -> getOutput2b()
20	CenterFrequencies	centerF[] - int	centerF[] -> getOutput1b()
21	CompressionThreshold_NL2	CT[] - double	CT[] -> getOutput1()
22	CompressionRatio_NL2	CR[] - double	CR[] -> getOutput1()
24	getMPO_NL2	MPO[] - double	MPO[] -> getOutput1()

26	GetMLE	MLE[] - double	MLE[] -> getOutput1()
27	ReturnValues_NL2	MAF[] - double BWC[] - double ESCD[] - double	MAF[] -> getOutput1() BWC[] -> getOutput2() ESCD[] -> getOutput3()
28	GetTubing_NL2	Tubing[] - double	Tubing[] -> getOutput1()
29	GetTubing9_NL2	Tubing[] - double	Tubing[] -> getOutput1()
30	GetVentOut_NL2	Ventout[] - double	Ventout[] -> getOutput1()
31	GetVentOut9_NL2	Ventout[] - double	Ventout[] -> getOutput1()
39	GetRECDh_indiv_NL2	RECDh[] - double	RECDh[] -> getOutput1()
40	GetRECDh_indiv9_NL2	RECDh[] - double	RECDh[] -> getOutput1()
41	GetRECDt_indiv_NL2	RECDt - double	RECDt -> getOutput1()
42	GetRECDt_indiv9_NL2	RECDt - double	RECDt -> getOutput1()