Jabra SDK for Android

Developer's Guide

Contents

1 Introduction

This document is an introduction to the SDK that enables developers to create Android apps and benefit from the features of selected Jabra products.

We expect the reader to be familiar with Android development in general. If you are new to Android development, we recommend you spend some time reading on developer.android.com before you start to work with the SDK.

Details on Jabra products can be found at www.jabra.com.

This document covers version A1.0.0-N1.0.12.0 of the SDK.

1.1 SDK purpose

The purpose of our Android SDK is to let you – a corporate or independent Android developer - develop apps that benefit from access to settings and other functionality in Jabra devices.

The scope of these apps is limited only by your imagination (and the functionality of the platform and our Jabra devices, of course).

The Android SDK is one member of the family of Jabra SDKs for desktop and mobile platforms. These SDKs offer very similar functionality, encapsulated and presented with APIs that are natural for each platform.

The list of supported Jabra devices may differ across platforms, mainly because of differences in connectivity (USB vs Bluetooth).

For an overview of our current platform support, please visit developer.jabra.com.

1.2 SDK scope

The SDK provides access to:

- Product images - suitable for display in the app.
- Settings – let your user control the settings of the connected Jabra device.
- Device buttons – some Jabra devices have 'application buttons' that your app can listen to. Use button events to trigger actions in your app.
- Device sensors – some Jabra devices have in-ear sensors that provide heart rate (pulse) and/or step rate. Your app can use this for health and activity monitoring.
- Call control assistance – if you are writing a softphone app, we have support for using buttons on the Jabra device to accept/reject a call.

Data for product images and settings are provided by the SDK accessing our servers, always using the latest information. This means that you can expect new Jabra devices to be supported 'automatically', without having to use an updated SDK, or releasing a new version of your app.

The current version of the Android SDK supports Jabra devices connected via Bluetooth.

1.2.1 Not in scope

If you are building a softphone app, you will know best how to connect the audio stream to the call provider you are using, the SDK does not attempt to control that.

However, as part of the demo app delivered with the SDK, we provide an example of how to establish an audio connection to a Jabra device – there are a few quirks in the Android APIs for this. Feel free to use it as you please.

1.3 Updates

Please check for updates at developer.jabra.com.

2 Overview of the SDK

2.1 Supported environments

2.1.1 OS versions

For the current version of the SDK, the settings are:

- minSdkVersion 19
- targetSdkVersion 25

As Android evolves, we will attempt to always let new SDK releases follow the most recent Android release as targetSdkVersion.

2.1.2 Hardware

The SDK contains a native library built for ARM and ARMV7 (which should also imply support for ARM64-v8a due to backwards compatibility). We currently do not support Mips or x86.

As all supported Jabra devices currently use Bluetooth to connect to the Android device, Bluetooth support is required.

The SDK requires an internet connection, as most device info is retrieved from our servers.

2.1.3 Connectivity

Currently, only Bluetooth is supported.

2.1.4 Other

The SDK does not depend on any UI properties (such as screen size, resolution).

The SDK requires JabraService version 1.7.1 or later to be installed on the Android device.

2.2 Compatibility with previous SDKs

For select partners, we have previously offered limited device connectivity through direct communication with JabraService.

The present SDK supersedes that.

The SDK still uses JabraService internally to communicate with Jabra devices, both because of the need for continued support for existing apps, and because JabraService has a strong coordinating role when using multiple apps.

2.3 Prerequisites

We strongly recommend using Android Studio, but if you have other preferences, that should also work. The SDK is just an .aar.

You will need an Android device. Using an emulator may work for initial UI creation and app functionality that does not depend on the SDK, but eventually you will need a device.

You will also need one or more Jabra devices. We recommend that you test with all devices that you plan to support in your app, as functionality may vary across devices.

2.4 Supported Jabra devices

At the time of writing, the following Jabra devices are supported by the Android SDK:

	Call control support	Busylight control	Button events	Sensor data
Jabra Evolve 65 Mono	Yes	Yes		
Jabra Evolve 65 Stereo	Yes	Yes		
Jabra Evolve 75	Yes	Yes		
Jabra Speak 810	Yes			
Jabra Speak 710	Yes		Yes	
Jabra Eclipse	Limited due to lack of device buttons.			
Jabra Motion	Yes			
Jabra Stealth	Yes			
Jabra Supreme	Yes			

Jabra Steel	Limited due to lack of button support in device.			
Jabra Elite Sport	Yes		Yes	RRI, HR, Step rate
Jabra Sport Pulse	Yes		Yes	HR, Step rate
Jabra Sport Coach	Yes		Yes	Step rate
Jabra Sport Pace	Limited due to lack of button support in device.		Yes	

All devices support retrieval of basic info (such as battery status, FW version).

See developer.jabra.com for updates.

2.5 Contents of the SDK

The SDK is a .zip file (downloadable after registration) from developer.jabra.com, containing:

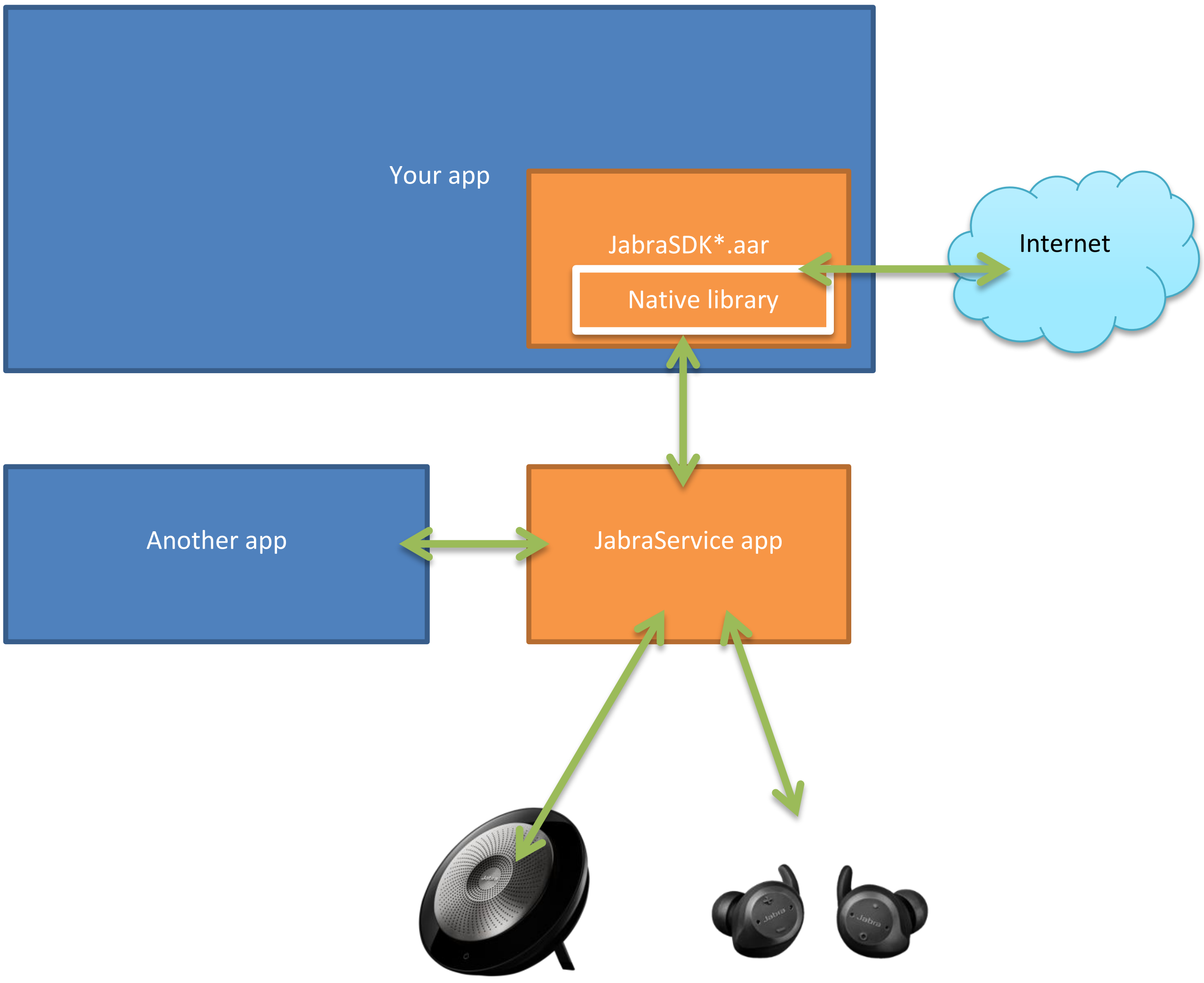
- This guide
- JabraSdk-x_y_z-Axyz-Na_b_c_d.aar
- JabraSdk-x_y_z-Axyz-Na_b_c_d-javadoc.jar
- JabraSdk-x_y_z-Axyz-Na_b_c_d-sources.jar
- Demo.zip – a full Android Studio project illustrating how to use the SDK.
    - Note that you will need your own API key if you want to build and run the project yourself – see below!
- App-debug.apk (runnable debug build of the demo app project)

2.6 Installation

Simply unpack the downloaded zip file to a folder of your choice.

2.7 Architecture

When using the .aar library in the SDK, your app is part of this overall architecture:



Your app uses the APIs of the JabraSDK library (which has a common cross-platform component inside) to communicate to the Jabra device(s) you want to use.

This happens through JabraService, but that dependency is not visible in the API.

The role of JabraService is to coordinate access from multiple apps (legacy, or using the SDK) to the same devices. Without it, only a single app would be able to communicate with the device.

3 Using the SDK

3.1 First steps

3.1.1 Download and unpack

Since you are reading this guide, you have probably already done that.

3.1.2 Study the Javadoc

We recommend that you browse through the Javadoc for the API. Keep it handy for later reference.

3.1.3 Run the demo app

Install and run the demo app (the apk is part of the download). Try it with different Jabra devices to get an impression of what the SDK can add to your app.

3.1.4 Study the demo app

Study the Android Studio project used to create the demo app. Feel free to change anything you want, but please note that you will need to obtain an API key of your own to actually run the demo app if you build and install it yourself.

Note that part of the demo app is an example of how audio can be collected from the microphone of the Jabra device, and how audio can be sent back to it. Part of that is a 'just-for-fun' pitch changer. This is simply to illustrate one way of handling audio routing; depending on your particular application that code example may be useful to you, or you can implement this by other means.

3.1.5 Get an API key

When you have decided to use the SDK, visit developer.jabra.com to obtain an API key for your app. This key identifies your app to our servers, and you probably want to file it outside of your source tree. The build.gradle file for the demo project illustrates one way to do this.

3.2 Building your own app

When you are ready to create your own app, feel free to copy+paste as you please from the source of the demo app.

3.2.1 Get your API key

You will need an API key for the SDK library to run. If the key is missing, the library will throw an exception. Visit developer.jabra.com to get an API key for your app.

3.2.2 Depend on the aar

In build.gradle for your app module, add a dependency to our JabraSDK*.aar.

Your can do this by adding .aar files for the scan of the libs folder (as the demo app does), or by adding a new module to your project, that just contains the .aar.

Note that we do not currently distribute the SDK via jcenter or similar methods; you will have to replace the .aar file manually when updates are available.

3.2.3 Connect to the SDK

When your app starts, you need to connect to the SDK. This is an asynchronous operation (as is most of the API).

In the demo app, this is handled by the DeviceConnector class, but you can implement this in other ways. The important point is that you do not have a reference to the SDK before the listener for *`JabraHelper.getJabraConnectionManagerInstance()`* has a successful result.

At that time, you probably want to set up a listener for the connection state of devices (as the demo app does).

3.2.4 Select a device

A common next step would be to identify devices of interest. *`JabraConnectionManager.getJabraDevices()`* does just that. Note that the list you get is a snapshot – if you pair new devices during the lifetime of your app, you probably want to get the list again.

The criteria for selecting a device (or several) is up to you, but a common candidate is the device which has an active voice link (Bluetooth HF profile).

You could of course also show a list of devices to the user, and let him or her pick one.

3.2.5 Connect to the device

After selecting one or more devices, you need to signal your interest in the device to the SDK. This is done by *`JabraDevice.connect()`*. Note that there is no corresponding disconnect() method. When you have signaled your interest in this way, the callback of the connect() method will receive the connection status of the device. When a device is connected, you can start using the JabraDevice API for the device.

Accidentally trying to talk to a disconnected device does no harm, but you will receive an error in the callback for what you're trying to do.

3.2.6 Communicate with the device

After establishing connection to a device, you can start talking to it. See the JavaDoc for the JabraDevice class (and the interfaces that it implements) for details.

Note that when connecting to a device, especially for the first time, some of the APIs make take a while to get back. This is especially true for the settings API, as it needs to talk to our servers and subsequently the device, before a full understanding of the settings of the device can be presented.

3.2.7 Settings

The intention with the settings API is to let you easily create a UI for all the settings of the device, without having to know the details of what a setting does. You simply create and maintain UI elements suitable for your app, and connect them to the settings data structures handled by the API. The JabraSettingsAdapter class in the demo is an example of how this can be done.

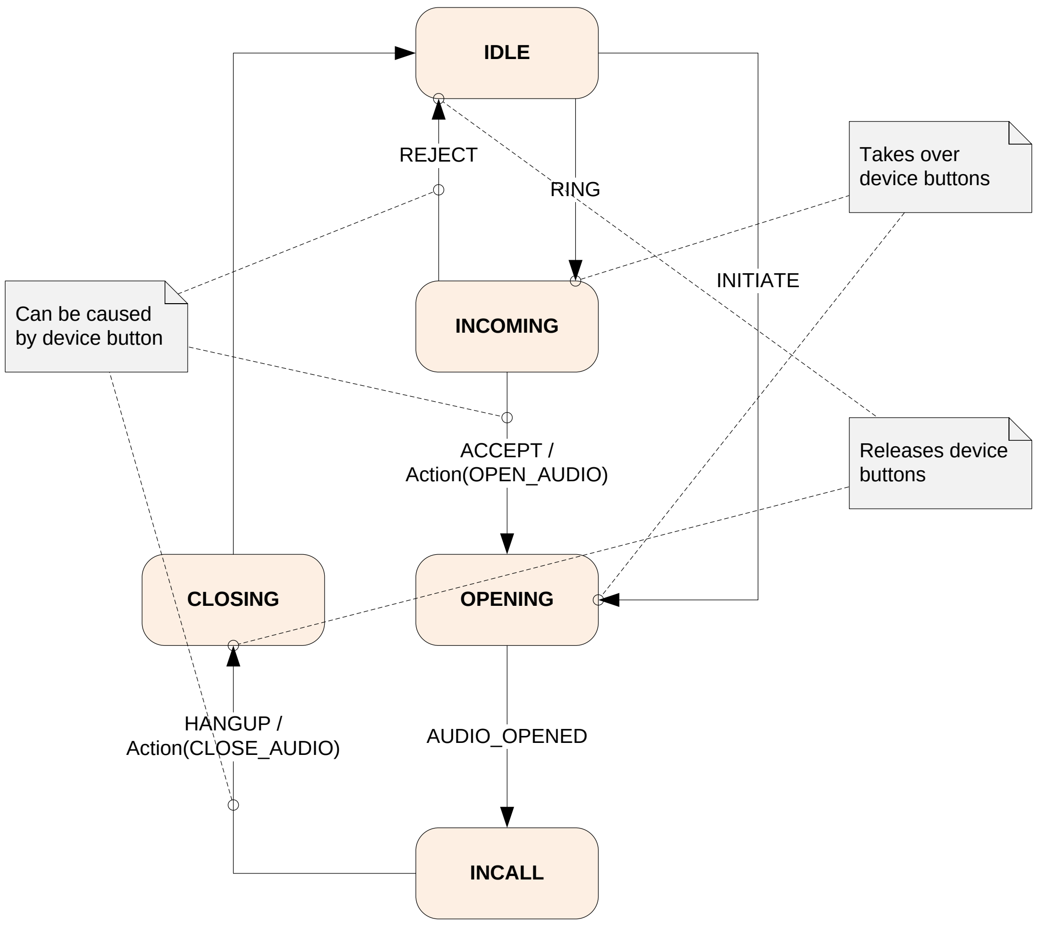
If you need to access a few specific settings for a device, these can be identified by their GUIDs in the collections of settings handled by the API. Please contact us through developer.jabra.com.

3.2.8 Using the CallControlHelper

If your app is a softphone, or if for other reasons you need to let the buttons of the Jabra device participate in audio routing, the CallControlHelper may be of use.

It is a fairly simple state machine that – with cooperation from your app – allows your user to use the device buttons in a way that is similar to what is experienced by using the device for a 'regular' phone call.

We have chosen not to let your app take over the buttons of the Jabra device without restrictions, as that can very easily interfere with the expected behavior of the Jabra device.


IDLE
REJECT
RING
Takes over device buttons
INITIATE
Can be caused by device button
INCOMING
Releases device buttons
ACCEPT / Action(OPEN_AUDIO)
CLOSING
OPENING
HANGUP / Action(CLOSE_AUDIO)
AUDIO_OPENED
INCALL

The CallControlHelper expects your app to feed it events (CallEvent). It will respond with info on its current state (CallState), and tell your app what actions it expects (CallAction). Internally, it will in some states, take over certain buttons of the device, to let the user accept/reject an incoming call as well as let the user end a call by using device buttons.

The button used will be the ones normally used for call control for a particular Jabra device.

Note that if another app has already taken over the buttons of the Jabra device (for example, if the Jabra Sport Life is running and connected to the same device), your app will not have access to the buttons of that device.

3.2.9 Getting help

As a registered SDK developer, you can use our support forum at developer.jabra.com