amazon alexa

Quick Start Guide

AVS Android Library

Version 1.0 | 19 May, 2017
Amazon Confidential

Table of Contents

This document is a draft and is subject to change.

AVS Android Library v1.0

This section provides a high-level overview of updates in v1.0.

New Features:

- Added the `AlexaUserSpeechListener`
- Provided access to active listening metadata
- Updated notifications UI to support interactions with Alexa
- Improved OOBE login/log out flow
- Added Local Stop, which immediately stops audio playback

Not Supported:

- Timers and alarms

Known Issues:

- When changing accounts, the first request following a login may fail.

Minimum Requirements

- Android Studio v2.3.1 or later
- JDK 8u121 or later

Overview

The Alexa Voice Service (AVS) Android Library allows OEMs to integrate Alexa into Android devices. The library is an AAR, and it contains an Android Service that exposes the following interfaces:

- `AlexaServices` - Is used to access all Alexa APIs. It contains three groups of functions for `Recongize`, `Settings`, and `Account`.
- `Recognize` - Is used to access all APIs for recognizing a user's speech. Functionality includes registering/deregistering the `AlexaStateListener and AlexaUserSpeechListener`, as well as starting/stopping the recording of speech.
- `Settings` - Is used to access all APIs related to Alexa settings. Functionality includes registering/deregistering the `AlexaSettingsListener`, as well as getting and setting `Locale`. `Locale` is used to change the language that Alexa will speak to the user in, and also determines the content that will be provided to the user.
- `Account` - Is used to manage the Amazon account used by `AlexaServices`. When you connect to AVS, we authenticate the application with the account. The full UX flow is built into the service, including a way to log out.

- `AlexaServicesConnection` - Manages the connection with `AlexaServices`. It must be created and connected before being used with any API operations. This interface includes the class `AlexaServicesConnection.ConnectionListener` which provides callbacks for connection updates to the `AlexaServices` interface. This interface also provides methods for getting connection status, registering/deregistering the `AlexaServicesConnection.ConnectionListener`, and connecting the `AlexaServicesConnection` to `AlexaServices`.
- `AlexaStateListener` - The listener that receives Alexa state updates.
- `AlexaSettingsListener` - Receives Alexa settings updates. It exposes a single method, `onLocaleChanged` that is called when Alexa's locale is changed.
- `AlexaUserSpeechListener` – The listener that receives user speech volume as it changes. This value is used to represent how well your application hears user speech while actively listening, and visualize the data for the user. For more information, see AVS UX Guidelines.

This library is designed to handle most interactions with AVS. Your main responsibility is to ensure that the user experience (UX) adheres to the rules put forth by Alexa States.

Alexa States

It is important that your application is aware of the `AlexaState` at the different stages of a user interaction. The `AlexaState` determines what interactions are valid/available to the user at a given time. The following is an enumeration of valid Alexa states:

- `IDLE` - Alexa is idle, and ready to listen (`Recognize`) for user speech.
- `PREPARING_TO_LISTEN` - Alexa is preparing to listen for user speech. No other interactions can be started while in this state.
- `LISTENING` - Alexa is listening for user speech. Closing the microphone or cancelling the operation to listen is a valid interaction while in a `LISTENING` state.
- `FINISHING_LISTENING` - Alexa is about to finish listening for user speech. No other interactions can be started while in this state.
- `THINKING` - Alexa is processing user speech. No other interactions can be started while in this state.
- `SPEAKING` - Alexa is speaking to the user. Interrupting Alexa with a new request, or canceling Alexa speech are valid interactions.
- `UNKNOWN` - Alexa should only be in an `UNKNOWN` state when not connected to `AlexaServices`.
- `ERROR` - This is a temporary state, and is only valid when something unexpected occurs that needs to be communicated to the user.

For additional information about Alexa States, see the included Javadocs.

How To Consume the AVS Android Client

The AVS Android Client is packaged as a Maven repository that includes the AAR, POM.xml (dependencies), and Javadoc. To consume `avsandroidclient-maven.zip`, follow these instructions:

1. Unzip the Maven repository.
2. Add a section to the root `build.gradle` repositories configuration. It should point to your local Maven repository. For example:

```
allprojects {
       repositories {
           maven {
               url "$ABSOLUTE_PATH_TO_UNZIPPED_REPOSITORY"
           }
           jcenter()
       }
}
```

3. Add a dependency to your `build.gradle` dependencies section:

```
dependencies {
       compile 'com.amazon.alexa:avsandroidclient:1.0.0'
}
```

4. Make sure the Maven repository is being consumed. You can use gradle to list your dependencies with this command: `gradle app:dependencies`

   IMPORTANT: This command assumes the module is named **app**.

Authentication Pre-Work

The AVS Android Library handles user authentication/user login with AVS, and meets the requirements and recommendations provided in the Alexa Voice Service UX Design Guidelines. There is some pre-work that you need to do before the library can handle authentication for your app:

1. Navigate to https://developer.amazon.com/login.html and login. If you don’t have an account, take this opportunity to create one.
2. From the top-nav, click **Alexa**, then locate Alexa Voice Service and click **Get Started >**.
3. In the upper-right locate **Register a Product** and select **Application** from the drop-down.
4. Follow the instructions in the registration wizard to create a new application. Make note of your **Application Type ID**, and make sure that you create an **API Key** for your application during **Security Profile** creation, you’ll need these soon.
5. After you complete registration, you’ll need to create two files locally: `api_key.txt` and `avs.properties`.

- Create `api_key.txt` in your project's assets folder and paste in your API Key. When finished, save.
- Create `avs.properties` in your project's assets folder and add your **Application Type ID**. The file should match this format, where `<<PRODUCT_ID>>` is your **Application Type ID**: `product_id=<<PRODUCT_ID>>`

`AlexaServicesConnection` will use this information during the authentication/login flow.

The `Account` API in the `AlexaServices` interface allows you to initiate the authentication flow or reach the sign out flow.

Amazon Account Management

The AVS Android Library completely handles Amazon account management for your application. Since a valid Amazon account is required for every interaction with Alexa, the library checks for an account each time an application connects via the `AlexaServiceConnection`. If an account is not found, the sign in flow is automatically started for the user on launch. This means that you do not need to worry about the logged in status of the user. Simply connect to `AlexaServices` when you are ready for the customer to interact with Alexa.

On connect, two `PendingIntents` (one for success, the other for failure) must be provided. The success `PendingIntent` should allow the user to begin interacting with Alexa immediately. The failure `PendingIntent` should return to your activity and reconnect to the `AlexaService` (triggering the log in flow).

Log out is also handled by the AVS Android Library. The `AlexaServices.Account` API provides access to the account management screen. While the user is signed in, calling this API will take the user to an Activity with a log out button.

How To Use the AVS Android Library

1. Create an `AlexaServicesConnection` when your Android application is started.
2. Call `AlexaServicesConnection.connect` `onStart` of your main Activity.
3. Call `AlexaServicesConnection.disconnect` `onStop` of your main Activity.
4. Use this connection with each of the `AlexaServices` interfaces/APIs.

For a full list of interfaces and associated methods, see the included JavaDoc.

Get Your First Response from Alexa

The following sample is intended to show the minimum amount of code required to get a working client of `AlexaServices` that can recognize user speech, and display Alexa state through a single button. This code also assumes that the app is given permission to use the microphone manually through Android Settings.

Sample Code

```
/*
 * Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 */

package com.amazon.alexa.avsandroidclient;

import android.Manifest;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v13.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.RelativeLayout;

import com.amazon.alexa.api.AlexaServices;
import com.amazon.alexa.api.AlexaServicesConnection;
import com.amazon.alexa.api.AlexaState;
import com.amazon.alexa.api.AlexaStateListener;

/**
 * This sample activity intends to show the minimum amount of code required
to get a working client
 * of AlexaServices which can recognize user speech, and display Alexa state
through a single
 * button. It also assumes that the application is given permission to use
the microphone manually
 * through Android Settings.
 */
public class SampleAlexaActivity extends AppCompatActivity
        implements AlexaServicesConnection.ConnectionListener,
AlexaStateListener {

    /**
     * The text for the button when it is ready to recognize.
     */
    private static final String RECOGNIZE_TEXT = "LISTEN";

    /**
     * The key name representing the logged in status of the user
     */
    private static final String LOGGED_IN = "loggedIn";

    /**
     * The action associated with the intent passed to the service. Actions
are required for
     * intent extras to be persisted.
     */
    private static final String LOGIN_ACTION = "LOGIN";
    /**
     * Simply a value to track asking for permission to record.
     */
    private static final int REQUEST_MICROPHONE = 1;

    /**
     * The AlexaServicesConnection to use when calling AlexaServices APIs.
     */
    private AlexaServicesConnection alexaServicesConnection;

    /**
     * The button to interact with Alexa.
     */
    private Button recognizeButton;

    @Override
    protected void onCreate(@Nullable final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Always create a single alexaServicesConnection during start up of
the application.
        alexaServicesConnection = new AlexaServicesConnection(this);

        // Create a user interface for the user to use.
        RelativeLayout layout = new RelativeLayout(this);

        recognizeButton = new Button(this);
        recognizeButton.setText(RECOGNIZE_TEXT);

        // Start recognizing user speech when clicked.
        recognizeButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(final View v) {
                if (!checkPermission(Manifest.permission.RECORD_AUDIO)) {
                    // Ensure we have permission to record for the sample
                    requestPermissions();
                } else {
                    // Start recognizing user speech
                    AlexaServices.Recognize.start(alexaServicesConnection);
                }
            }
        });
        RelativeLayout.LayoutParams centeringParams =
                new RelativeLayout.LayoutParams(
                        ViewGroup.LayoutParams.WRAP_CONTENT,
                        ViewGroup.LayoutParams.WRAP_CONTENT);
        centeringParams.addRule(RelativeLayout.CENTER_IN_PARENT,
RelativeLayout.TRUE);
        layout.addView(recognizeButton, centeringParams);
        this.setContentView(layout);
    }

    /**
     * Always connect the AlexaServicesConnection in onStart.
     *
     * Ensure all AlexaServicesConnection.ConnectionListeners are registered
before
     * connecting. Listeners must be registered to know when AlexaServices
APIs can be called,
     * as the AlexaServicesConnection is not guaranteed to be connected
immediately
     * after AlexaServicesConnection#connect(PendingIntent) is called.
     */
    @Override
    protected void onStart() {
        super.onStart();
        if (!checkPermission(Manifest.permission.RECORD_AUDIO)) {
            requestPermissions();
        }

        Intent successCallback = new Intent(this, SampleAlexaActivity.class);
        successCallback.setAction(LOGIN_ACTION);
        successCallback.putExtra(LOGGED_IN, true);
        Intent failureCallback = new Intent(this, SampleAlexaActivity.class);
        failureCallback.setAction(LOGIN_ACTION);
        failureCallback.putExtra(LOGGED_IN, false);
        PendingIntent loggedInCallback = PendingIntent.getActivity(this, 1,
successCallback, 0);
        PendingIntent loggedOutCallback = PendingIntent.getActivity(this, 2,
failureCallback, 0);
        alexaServicesConnection.registerListener(this);
        alexaServicesConnection.connect(loggedInCallback, loggedOutCallback);
    }

    @Override
    protected void onNewIntent(Intent intent) {
        setIntent(intent);
        boolean isLoggedIn = intent.getBooleanExtra(LOGGED_IN, false);
    }

    /**
     * Always disconnect the AlexaServicesConnection in onStop.
     */
    @Override
    protected void onStop() {
        super.onStop();
        alexaServicesConnection.disconnect();
        alexaServicesConnection.deregisterListener(this);
    }

    /**
     * The AlexaServicesConnection is connected, and AlexaServices APIs can
now be used. This is the
     * best place to register listeners, such as the AlexaStateListener for
AlexaState changes,
     * which is most important interface to implement as an AVS Android
Library client.
     */
    @Override
    public void onConnected() {
        AlexaServices.Recognize.registerListener(alexaServicesConnection,
this);
```

```
    }

    /**
     * The AlexaServicesConnection is disconnecting. This is the ideal place
to deregister listeners.
     */
    @Override
    public void onDisconnected() {
        AlexaServices.Recognize.deregisterListener(alexaServicesConnection,
this);
    }

    /**
     * The AlexaState has changed to a new value, most likely due to having a
dialog with a user.
     * Each state has a specific requirement on what actions the user is
capable of taking. The
     * implementation here is the simplest correct implementation, but does
not take advantage
     * of all possible interactions (such as allowing the user to 'barge in'
on Alexa when the state
     * is AlexaState.SPEAKING). For a full description of behaviors visit:
     *
     *     https://developer.amazon.com/public/solutions/alexa/alexa-voice-
service/content/alexa-voice-service-ux-design-guidelines#understand
     */
    @Override
    public void onAlexaStateChanged(final AlexaState alexaState) {
        if (alexaState == AlexaState.IDLE) {
            recognizeButton.setText(RECOGNIZE_TEXT);
            recognizeButton.setEnabled(true);
        } else {
            recognizeButton.setText(alexaState.toString());
            recognizeButton.setEnabled(false);
        }
    }

    /// Helpers for getting recording permissions
    private boolean checkPermission(final String permission) {
        return ContextCompat.checkSelfPermission(this, permission) ==
                PackageManager.PERMISSION_GRANTED;
    }

    private void requestPermissions() {
        ActivityCompat.requestPermissions(
                this,
                new String[]{Manifest.permission.RECORD_AUDIO},
                REQUEST_MICROPHONE);
    }
}
```

Release Notes

Version	Date	Notes
v1.0	5/19/2017	Added the `AlexaUserSpeechListener` API, provided access active listening metadata, updated notifications to support interactions with Alexa, improved OOBE login/log out flow, added Local Stop for audio playback, and general bug fixes.
v0.2	4/29/2017	Added support for the AudioPlayer and Speaker interfaces, and ProGuard, improved OOBE for permissions, and reduced battery usage while location services are enabled.
v0.1	4/13/2017	Added instructions for consuming local Maven repositories.
v0.1	4/6/2017	Initial release of the Alexa Voice Service Android Library. Includes AAR, Quick Start Guide, and Javadocs.