Zello Android client SDK

Overview

The Zello Android client SDK allows you to integrate Zello Work push-to-talk into your own application. The SDK uses cross-process communication to let your app connect to the Zello Work app installed on the device and remotely control it. Supported features include:

- Send voice messages
- Get notifications about incoming voice messages
- Get the list of contacts and their status
- Configure and switch user accounts
- Connect and disconnect channels
- Mute and unmute users or channels
- Set availability status
- Set custom text status
- Control auto-run and other Zello app options

Current Version

The stable release for the Zello Work Android SDK is v4.100.3.

Installation

Sign up for Zello Work account

Go to http://zellowork.com/ and click **Start your network** button. If you already have a network, click **Sign In**. A free Zello Work account supports up to five users and has no time limit.

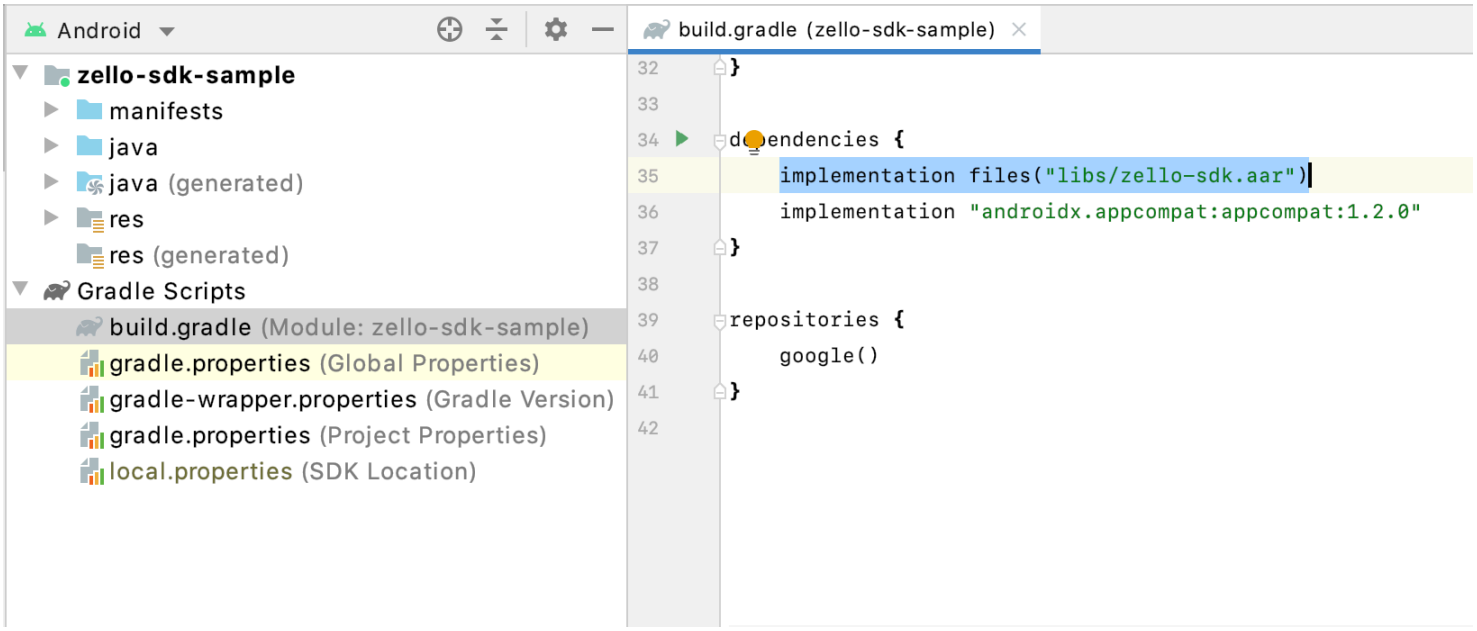
Get Zello Work app

Before you can use the SDK, you must install the Zello app on your phone. You can do this by getting the Zello app from Google Play, by downloading the Zello Work app from the **Get app** section of the web console or by navigating to `http://<network name>.zellowork.com/app` on your phone.

Install Android Studio and configure your project

Download Android Studio and install it. Open your existing project or create a new one. The minimum API level supported by the SDK is 15 (Ice Cream Sandwich).

Place zello-sdk.aar file into `libs` folder of your project, then edit the gradle.build file to include the new AAR dependency: `implementation files("libs/zello-sdk.aar")`.



Using the SDK

Configuring the SDK

The first thing you need to do in your app to start using Zello SDK is to configure it. In the most cases you'd want to do it in your `Application.onCreate()` method:

```
public class App extends Application {

   @Override
   public void onCreate() {
       super.onCreate();

       Zello.getInstance().configure(this);
   }

}
```

This will automatically select the Zello app to connect to. Alternatively, you can provide a specific package name:

```
public class App extends Application {

   @Override
   public void onCreate() {
       super.onCreate();

       Zello.getInstance().configure("com.loudtalks", this);
   }

}
```

Here `com.loudtalks` is the package name of Zello app. `net.loudtalks` can be used to connect to Zello Work app instead.

Sending voice messages

To start a voice message to the currently selected contact, call `Zello.getInstance().beginMessage()`. To stop sending the message, call `Zello.getInstance().endMessage()`. Here is a snippet of how to make a push-to-talk button in your activity:

```
Button pttButton = (Button)findViewById(R.id.pttButton);
pttButton.setOnTouchListener(new View.OnTouchListener() {
   @Override
   public boolean onTouch(View v, MotionEvent event) {
      int action = event.getAction();
      if (action == MotionEvent. ACTION_DOWN ) {
         Zello.getInstance().beginMessage();
      } else if (action == MotionEvent. ACTION_UP || action == MotionEvent. ACTION_CANC
         Zello.getInstance().endMessage();
      }
   return false;
}
});
```

To successfully send a message, one needs to select a contact first. The SDK includes a built-in activity that you can display to let user select a contact:

```
Zello.getInstance().selectContact("Select a contact", new Tab[]{Tab.RECENTS, Tab.USER
S, Tab.CHANNELS}, Tab.RECENTS, Theme.DARK);
```

You can also select a contact programmatically:

```
Zello.getInstance().setSelectedUserOrGateway("test"); // selects a user with username
 "test"
```

Handling Zello SDK events

The Zello SDK contains an events interface which you can implement to be notified about changes in incoming and outgoing messages, state, app online status, sign in progress etc. In most cases, your implementation will be a part of your activity code.

```
public class MyActivity extends Activity implements com.zello.sdk.Events {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Zello.getInstance().subscribeToEvents(this);
   }

   @Override
   protected void onDestroy() {
      super.onDestroy();
      Zello.getInstance().unsubscribeFromEvents(this);
   }

   // Events interface implementation
   @Override
   void onAppStateChanged(){}

   @Override
   void onAudioStateChanged(){}

   @Override
   void onContactsChanged(){}

   @Override
   void onMessageStateChanged(){}

   @Override
   void onSelectedContactChanged(){}

   @Override
   void onLastContactsTabChanged(){}

   @Override
   void onMicrophonePermissionNotGranted(){}

   // ...
}
```

> **NB**: All events interface methods are called on **UI thread**, so if you need to do any potentially slow processing, move it to background thread.

Switching user accounts

If the Zello Work app already has a user account configured and signed in, the SDK will connect to the existing user session so no repeat sign in is necessary. When needed, you can programmatically sign in Zello to the desired user account or sign out to stop the active session:

```
Zello.getInstance().signOut(); // Signs out the current user
Zello.getInstance().signIn("mynetwork", "myuser", "mypassword"); // Signs in into "my
network" network as "myuser"
```

Both `signIn` and `signOut` are asynchronous. Subscribe for Zello SDK events and implement `Events.onAppStateChanged()` to be notified about sign in progress or errors:

```
@Override
void onAppStateChanged(){
   Zello.getInstance().getAppState(_appState);

   Error error = null;
   String state = "";
   boolean showCancel = false, cancelEnable = true;

   if (!_appState.isAvailable()) {
        state = "Zello Work app is not installed";
   } else if (_appState.isInitializing()) {
        state = "Connecting to the Zello Work app...";
   } else if (_appState.isConfiguring()) {
        state = "Configuring Zello Work app...";
   } else if (!_appState.isSignedIn()) {
        if (_appState.isSigningIn()) {
            state = "Signing in...";
            showCancel = true;
            cancelEnable = !_appState.isCancellingSignin();
        } else if (_appState.isSigningOut()) {
            state = "Signing out...";
        } else if (_appState.isWaitingForNetwork()) {
            error = _appState.getLastError();
            state = "Waiting for network connection";
            showCancel = true;
        } else if (_appState.isReconnecting()) {
            error = _appState.getLastError();
            state = "Reconnecting in %seconds%...".replace("%seconds%", NumberFormat.ge
            showCancel = true;
        } else {
            state = "Signed out";
        }
    }
}
```

> **NB**: `Zello.getAppState(AppState)` and similar methods write a snapshot of the requested state into the provided object. Afterwards, the object state remains "frozen" (even if the application state changes) and **will not** update automatically. To get fresh data, call `Zello.getAppState(AppState)` again.

Battery life optimization

You can improve your apps power efficiency and reduce data usage by telling the Zello SDK when your app

switches to the background or the user leaves the screen showing the Zello UI. You can do this by calling `Zello.getInstance().enterPowerSavingMode()`. When in power saving mode, the Zello Work app limits communication to the server and postpones any non-critical updates. It doesn't affect your ability to send or receive messages. Make sure to call `Zello.getInstance().leavePowerSavingMode()` when the Zello UI reappears on the screen.

`Activity.onPause()` and `Activity.onResume()` are good places to call these methods:

```
public class MyActivity extends Activity {

   @Override
   protected void onPause() {
      super.onPause();
      Zello.getInstance().enterPowerSavingMode();
   }

   @Override
   protected void onResume() {
      super.onResume();
      Zello.getInstance().leavePowerSavingMode();
   }
```

When your app no longer needs the SDK, call `Zello.getInstance().unconfigure()` to release resources.

Going live with your Zello-enabled app or service

All apps using Zello SDK must adhere to the following:

- All UI screens, embedding the Zello SDK must include the Zello logo
- Use the Zello logo and "Zello" name, when referencing Zello-powered features inside of your app or service
- Send us the app for approval before distributing to any third parties or customers

Additional resources

Zello SDK samples

Sample	Description
zello-sdk-sample	Master sample, showing all features available in the SDK
zello-sdk-sample-signin	Signing in and out
zello-sdk-sample-ptt	Sending voice messages
zello-sdk-sample-contacts	Working with the contact list
zello-sdk-sample-misc	Advanced SDK options and settings

Documentation

- Zello SDK reference
- Zello SDK migration guide (for legacy SDK users)
- Zello Work server API

See also

- Zello Work server API libraries