Zello Android client SDK

Overview

Zello Android client SDK allows you integrate Zello for Work push-to-talk into your own application. The SDK uses cross-process communication to let your app connect to Zello 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

Installation

Sign up for Zello for 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 for Work account supports up to five users and has no time limit.

Get Zello for Work app

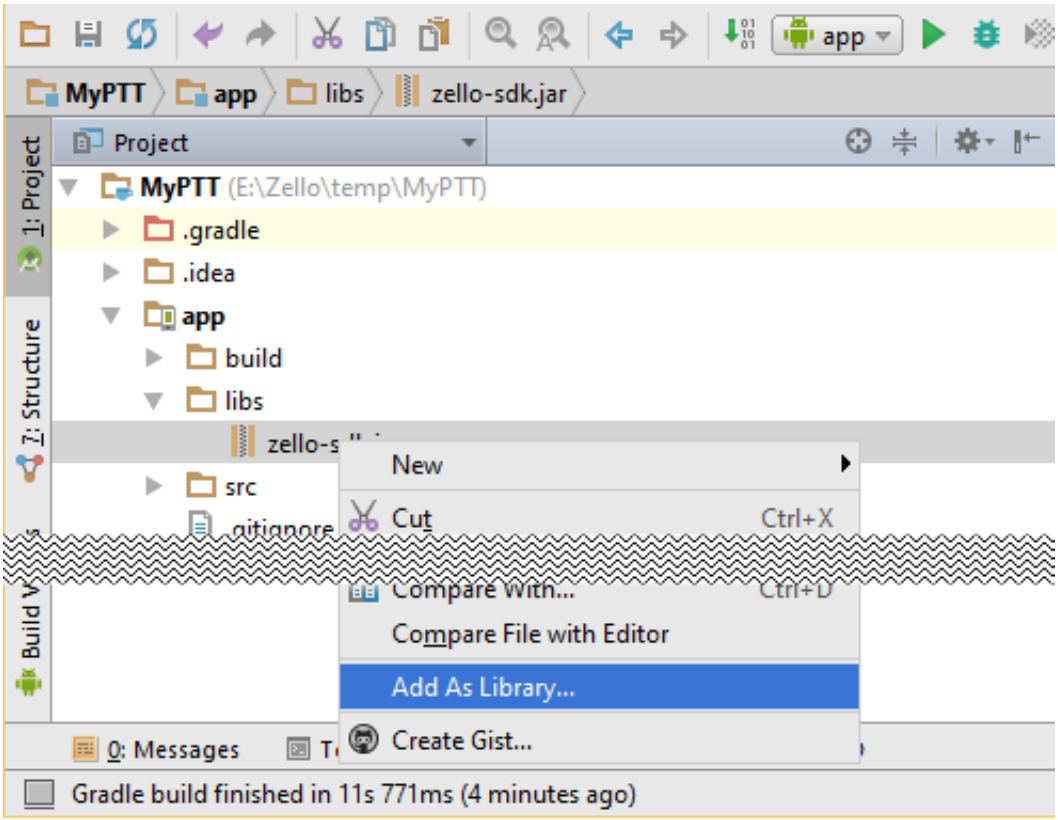
Before you can use the SDK install Zello for Work app on your phone. You can do it from **Get app** section of the web console or by navigating to `http://<network name>.zellowork.com/app` on your phone.

> **NB**: Zello app downloaded from Google Play is not supported by the SDK.

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 4 (Donut).

Place zello-sdk.jar file into `libs` folder of your project, then right-click the file in Android Studio and select “Add as Library…”.



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("net.loudtalks", this);
    }

}
```

Here `net.loudtalks` is the package name of Zello for Work app.

Sending voice messages

To start a voice message to 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_CANCEL ) {
         Zello.getInstance().endMessage();
      }
   return false;
}
});
```

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

```
Zello.getInstance().selectContact("Select a contact", new Tab[]{Tab.RECENTS, Tab.USERS, Tab.CH
```

You can also select a contact programmatically:

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

Handling Zello SDK events

Zello SDK supports 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 the 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(){}

   // ...
}
```

> **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 Zello 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 "mynetwork" n
```

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 for Work app is not installed";
   } else if (_appState.isInitializing()) {
        state = "Connecting to the Zello app...";
   } else if (_appState.isConfiguring()) {
        state = "Configuring Zello 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.getInstan
            showCancel = true;
        } else {
            state = "Signed out";
        }
    }
}
```

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

Battery life optimization

You can improve your app power efficiency and reduce data usage by telling Zello SDK when your app switches to background or user leaves the screen showing Zello UI. You do this by calling `Zello.getInstance().enterPowerSavingMode()`. When in power saving mode Zello app limits communication to the server postponing any non-critical updates. It doesn't affect your ability to send or receive messages. Make sure to call `Zello.getInstance().leavePowerSavingMode()` when Zello UI appears 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. Most apps should do it in `Application.onTerminate()`:

```
public class App extends Application {

   @Override
   public void onTerminate() {
      super.onTerminate();
      Zello.getInstance().unconfigure();
   }

}
```

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 contact list
zello-sdk-sample-misc	Advanced SDK options and settings

Documentation

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