Zello SDK Overview

The Zello SDK for Android is an easy and powerful way to interact with the Zello API. The SDK works by providing an interface that interacts with the Zello for Work app installed on your device to send and receive messages.

Installation:

The first step is to install the Zello for Work app onto your device. This is required because the SDK interacts with this app to send and receive voice messages.

Next, download Android Studio and create the project you want to integrate the Zello SDK into. The target of the project must be at least API 4 (Donut).

Lastly, download the zello-sdk.jar file and drag it into the libs folder of your project. Then, right-click the file in Android Studio and select “Add as Library…”.

Configure the Zello SDK:

The *Zello* class acts as the primary means of interaction with the Zello SDK. The *Zello* class contains most of the methods you will need to create your Push-To-Talk (PTT) app. There are two methods of initialization for the Zello SDK.

1. public static void initialize(String, Context)
2. public static void initialize(String, Context, Events)

The first method initializes the SDK without immediately subscribing to *Events*. Conversely, the second method initializes the SDK and immediately subscribes to *Events*. An example for the first method of initialization would be to initialize the SDK in the onCreate() method of the Application.

```
public class App extends Application {

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

    Zello.initialize("com.pttsdk", this);
  }

}
```

In this method of initialization, we are delegating responsibility to the *Activities* to subscribe to *Events* through the Zello SDK. An activity can do this by calling the subscribeToEvents() method through the *Zello* class.

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

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

     Zello.subscribeToEvents(this);
  }

  // ... Implement Events Methods
```

NOTE: Don't forget to add the *Application* name in your AndroidManifest.xml.

An example of the second method of initialization would be in a standalone *Activity* that would be utilizing the Zello SDK. This is the means of initialization in our sample projects, as these are very simple apps.

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

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

     Zello.initialize("com.pttsdk", this, this);
  }

  // ... Implement Events Methods
```

Sending Messages:

The *Zello* class provides two key methods for voice message communication.

1. beginMessage()
2. endMessage()

The beginMessage() method begins the voice message while the endMessage() method ends the voice message. A standard example for the use of these methods would be to have a *Button* that the user presses down to begin the message and releases to end the message.

```
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.beginMessage();
     } else if (action == MotionEvent.ACTION_UP || action == MotionEvent.ACTION_CANCEL) {
        Zello.endMessage();
     }

     return false;
  }
});
```

NOTE: Before the invocation of beginMessage(), there must be a selectedContact to send the message to.

Any message update will invoke the onMessageStateChanged() *Events* method that your *Activity* is implementing. You can receive information about this message update by using the *Zello* class.

```
@Override
public void onMessageStateChanged() {
  Zello.getMessageIn(messageIn);
  Zello.getMessageOut(messageOut);

  boolean incoming = messageIn.isActive(); // Is incoming message active?
  boolean outgoing = messageOut.isActive(); // Is outgoing message active?

  if (incoming) {
     // … Code for handling incoming message
  } else if (outgoing) {
    // … Code for handling outgoing message
  }
}
```

Zello SDK Lifecycle:

Just like an *Activity*, the Zello SDK has lifecycle methods that should be invoked to conserve battery and minimize data usage. These methods work by changing the status of communication between the Zello for Work app and the Zello SDK. There are three lifecycle methods that the *Zello* class contains.

1. pauseZelloUpdates()
2. resumeZelloUpdates()
3. killZelloUpdates()

When your app doesn't need immediate communication with the Zello for Work app, invoke the pauseZelloUpdates() method. When communication is needed again, invoke the resumeZelloUpdates() method.

```
private void doBusyWork() {
  Zello.pauseZelloUpdates();

  int i = 0;
  while (i < Integer.MAX_VALUE) {
    i++;
  }

  Zello.resumeZelloUpdates();
}
```

When the Zello SDK is no longer needed, invoke the killZelloUpdates() method. This could happen at either the *Application* lifecycle or the *Activity* lifecycle depending on how you initialized the Zello SDK.

```
public class App extends Application {

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

    Zello.killZelloUpdates();
  }

}

public class MiscActivity extends Activity implements com.zello.sdk.Events {

  @Override
  protected void onDestroy() {
    super.onDestroy();

    Zello.killZelloUpdates();
  }
```

The Zello SDK also has an *AppState* class to retrieve the state of the Zello SDK at any given moment. This *AppState* class can be used to retrieve key lifecycle information about the SDK and the authentication status of a user. To retrieve this information, call the getAppState() method on the *Zello* class.

```
private boolean userIsSignedIn() {
   com.zello.sdk.AppState appState = new com.zello.sdk.AppState();
   Zello.getAppState(appState)

   return appState.isUserSignedIn();
}
```

Handling Zello Events:

The Zello SDK uses the *Events* interface to communicate changes between the Zello for Work app and your PTT app. To start listening to *Events*, have your *Activity* implement the *Events* interface and subscribe to *Events* through the *Zello* class.

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

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

     Zello.subscribeToEvents(this);
  }

  // ... Implement Events Methods
```

Then, implement the methods defined in the *Events* interface.

NOTE: The *Events* methods are invoked on the UI thread.

When your *Activity* no longer needs to listen to *Events*, unsubscribe from *Events* using the unsubsubscribeFromEvents() method.

```
  @Override
  protected void onDestroy() {
     super.onDestroy();

     Zello.unsubscribeFromEvents(this);
  }
```

Authentication:

Before messages can be sent and received, the Zello SDK expects a user to be authenticated with the network. If your Zello for Work account is already configured, reauthentication is NOT necessary, as the Zello app will automatically attempt authentication with the network using these credentials.

The *Zello* class contains two methods to authenticate a user with the network.

1. public boolean signIn(String, String, String)
2. public boolean signIn(String, String, String, boolean)

Any authentication update will invoke the onAppStateChanged() *Events* method that your *Activity* is implementing. You can use this method to retrieve the most recent *AppState* and determine the authentication status.

The *Zello* class also contains two other authentication methods.

1. public void signOut()
2. public void cancelSignIn()

The signOut() method unauthenticates the user from the network while the cancelSignIn() method cancels the ongoing authentication request.

Sample Projects:

Provided with the Zello SDK are five sample projects illustrating proper use of the Zello SDK. The Zello SDK Sample project is an in-depth illustration on how to integrate every piece of the Zello SDK into one PTT app from start to finish. This project is a great example of how to build your app using the SDK.

The next four tutorials are provided in a tutorial fashion. Each project depends on the successful completion of the previous project in the list. The order to complete these tutorials is:

1. Zello SDK Sample Sign In
2. Zello SDK Sample Contacts
3. Zello SDK Sample PTT
4. Zello SDK Sample Miscellaneous

Each sample project provides a clear example of how to interact with the Zello SDK for specific purposes. The Sign In project contains all of the methods related to

authentication. The Contacts project contains all of the methods related to displaying and selecting a *Contact* for the user to communicate a voice message with. The PTT project contains all of the methods related to message sending, receiving and audio. Lastly, the Miscellaneous project contains all of the methods that don’t fall into the above categories, such as locking and unlocking the app and setting the status.