Amelia SDK 3.0.0

Prerequisites

This guide assumes Android Studio 2.X and Gradle 3.3 is used as build environment.

Setup

First, add below code into your project's `proguard-rules.pro` file:

```
-keepattributes *Annotation*
-keepattributes SourceFile,LineNumberTable
-keep public class * extends java.lang.Exception
-dontwarn java.nio.file.Files
-dontwarn java.nio.file.Path
-dontwarn java.nio.file.OpenOption
-dontwarn org.codehaus.mojo.animal_sniffer.IgnoreJRERequirement
```

Next,place `ameliasdk-3.0.0.aar` in a folder named 'libs' in your module and update the module's `build.gradle` with:

```
apply plugin: 'com.android.application'

repositories {
  //...
  flatDir {
    dirs 'libs'
  }
}

//...
dependencies {
  //...
  compile 'com.squareup.okhttp3:okhttp:3.6.0'
  compile 'com.squareup.okhttp3:okhttp-urlconnection:3.6.0'
  compile 'net.ipsoft.amelia.sdk:ameliasdk:3.0.0@aar'
}
```

Finally add `android.permission.INTERNET` to your `AndroidManifest.xml`.

Starting a conversation

Follow these steps to start a new conversation with an anonymous user:

Step 1. Configure the AmeliaChat instance:

```
AmeliaChatBuilder builder = new AmeliaChatBuilder()
    .setBaseUrl("<AMELIA_BASE_URL>")
    .setAllowAnonymous(true)
    .setDomainSelectionMode(DomainSelectionMode.manual);
```

Step 2. Register a listener to act on session/conversation life cycle events, extend `BaseSessionListener` as opposed to `ISessionListener` to only override methods relevant to you. See `ISessionListener` JavaDoc for a complete set of callbacks.

```
builder.addSessionListener(new BaseSessionListener() {

    @Override
    public void onDomainSelectionRequired(List<Domain> domains) {
        ameliaChat.selectDomain(domains.get(0));
    }

    @Override
    public void onConversationStart() {
```

```
            Toast.makeText(context, "Conversation started!", Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onSessionFail(AmeliaError error) {
            Toast.makeText(context, error.message, Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onDomainFail(AmeliaError error) {
            Toast.makeText(context, error.message, Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onConversationFail(AmeliaError error) {
            Toast.makeText(context, error.message, Toast.LENGTH_SHORT).show();
        }
    });
```

Step 3. Register a listener to act on chat events, extend `BaseConversationListener` as opposed to `IConversationListener` to only override methods relevant to you. See `IConversationListener` JavaDoc for a complete set of callbacks.

```
    builder.addConversationListener(new BaseConversationListener() {
        @Override
        public void outboundFinalTextMessage(AmeliaOutboundMessage body) {
            Toast.makeText(context,
                    body.getFromUserDisplayName() + " says: " + body.getMessageText(),
                    Toast.LENGTH_SHORT)
                    .show();
        }
    });
```

Step 4. Create the AmeliaChat instance:

```
    ameliaChat = builder.build();
```

Step 5. Start the conversation:

```
    ameliaChat.startNewConversation();
```

This will start the sequence of events required to start a new conversation and you should see toasts similar to this:

```
Conversation started!
```

```
Amelia says: My Greetings Anonymous User!
```

Login methods

If anonymous logins are disabled via `AmeliaChatBuilder.setAllowAnonymous(false)` a call to `ISessionListener.onLoginRequired` will come as a result of an attempt to `IAmeliaChat.startNewConversation()`. The list of auth systems should be used to assign an auth system to the `LoginOptions` that are passed to `IAmeliaChat.login(LoginOptions)`.

When anonymous logins are enabled, login can be triggered via `IAmeliaChat.stepupLogin()`, which will return `ISessionListener.onLoginRequired(List<AuthSystems>)`.

There are two classes of auth systems, internal and external. Currently, external logins are based on SAML.

Internal login

Internal login will have it's `AuthSystem.code` set to "internal". To login with an internal auth system, simply assign username / password to the `LoginOptions` instance.

SAML login

SAML login require a little extra work on the user of the SDK. The `AuthSystem.getLoginPath()` holds a URL that needs to be loaded in a WebView where a session will be established. When login is complete, the SDK will extract the session cookie from the app's WebView cookie store. Alternatively, clients may pass the cookie directly to `LoginOptions`.

Please refer to the accompanying sample app for a reference implementation.

Voice playback

Voice playback is enabled by default. To completely disable voice playback you may pass in `SpeechParams` as:

```
new AmeliaChatBuilder()
    .setBaseUrl("<AMELIA_BASE_URL>")
    .setSpeechParams(new SpeechParams(true))
    .build();
```

At runtime voice playback is muted/unmuted with `IAmeliaChat.mute()` and `IAmeliaChat.unmute()`.

Speech recognition

The SDK does not provide an API for speech to text as there are platform APIs that supports speech recognition to great effect. The entry point is `android.speech.SpeechRecognizer` and a sample implementation follow this setup:

```
SpeechRecognizer speechRecognizer = SpeechRecognizer.createSpeechRecognizer(getContext());
speechRecognizer.setRecognitionListener(new RecognitionListener() {
    // respond to speech callbacks
});

Intent recognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
recognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM
recognizerIntent.putExtra(RecognizerIntent.EXTRA_PARTIAL_RESULTS, true);
recognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, domain.getLocaleLanguageTag());

speechRecognizer.startListening(recognizerIntent);
```

Note that you may want to pass in the `Domain.getLocaleLanguageTag()` to the recognizer `Intent` to recognize the appropriate language. Also note that the speech recognizer requires permission `android.permission.RECORD_AUDIO`.

MMO Download

MMO downloads are reported on the `IConversationListener` interface as an `outboundMmoDownloadMessage(...)`. The passed `DownloadMessage` contains necessary information to download associated files. Please note: it is necessary to check the existence of `getMetadata()` as it may be `null` in case it wasn't fetched properly, e.g. due to a network error. If it doesn't exist, you may retry with the asynchronous `fetchMetadata()` method. Sample code to listen to and act on `outboundMmoDownloadMessage(...)`:

```
builder.addConversationListener(new BaseConversationListener() {
    @Override
    public void outboundMmoDownloadMessage(AmeliaOutboundMessage ameliaOutboundMessage) {
        DownloadMessage downloadMessage = ameliaOutboundMessage.getDownloadMessage();
        if (downloadMessage.getMetadata() != null) {
            MmoMetadata mmoMetadata = downloadMessage.getMetadata();
            if (mmoMetadata.getValue().size() > 0) {
                downloadMessage.download(0, downloadListener, context);
            }
        }
    }
});
```

Here, the `downloadListener` may be implemented as follows:

```
final DownloadMessage.IDownloadListener downloadListener = new DownloadMessage.IDownloadListener() {

    @Override
    public void onDownloadFailed(AmeliaError error) {
        Toast.makeText(MainActivity.this, "Download failed", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDownloadSuccess(DownloadedMmo mmo) {
        Toast.makeText(MainActivity.this, "Downloaded: " + mmo.uri, Toast.LENGTH_SHORT).show();
    }
};
```

MMO Upload

MMO file uploads are requested on the `IConversationListener` interface via `onUploadRequest(...)`. The `fileType` argument is used to select a file of appropriate type. E.g.:

```
builder.addConversationListener(new BaseConversationListener() {
    @Override
    public void onUploadRequest(AmeliaOutboundMessage ameliaOutboundMessage) {
        Toast.makeText(MainActivity.this, "Upload requested", Toast.LENGTH_SHORT).show();
        openFileChooser(ameliaOutboundMessage.getFromUserDisplayName(), ameliaOutboundMessage.getUploadMes
    }

    @Override
    public void onUploadSuccess(String fromUserDisplayName, String fileName, String url) {
        Toast.makeText(MainActivity.this, "Upload succeeded", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onUploadFailed(String fromUserDisplayName, String fileName, String fileType) {
        Toast.makeText(MainActivity.this, "Upload failed", Toast.LENGTH_SHORT).show();
    }
});
```

Typically, `openFileChooser` would pass out an `Intent.ACTION_GET_CONTENT` to let the user select a file for upload:

```
private void openFileChooser(String fromUserDisplayName, String fileType) {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.addCategory(Intent.CATEGORY_DEFAULT);
    intent.setType("*/*");
    Intent i = Intent.createChooser(intent, fromUserDisplayName  + " requested " + fileType);
    startActivityForResult(i, FILE_REQUEST_CODE);
}
```

Then, the result would be picked up and posted back to Amelia via `IAmeliaChat.uploadFile(...)`:

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case FILE_REQUEST_CODE:
            if (resultCode == Activity.RESULT_OK) {
                ameliaChat.uploadFile(data.getData(), this);
            }
            break;
    }
}
```

Forms

The `AmeliaOutboundMessageAttributes` on a received `AmeliaOutboundMessage` may contain a non-null value `FormInputData`. This means that there is a form to display in the client.

The client is expected to provide a user interface for displaying the form and post back the result using `IAmeliaChat.submitForm(...)` or by simply replying with the selected value in a chat message (`IAmeliaChat.say(...)`).

The allowed methods for submitting are determined by the value of `FormInputData.getAllowedUserInputs()`

```
builder.addConversationListener(new BaseConversationListener() {

  @Override
  public void outboundFinalTextMessage(AmeliaOutboundMessage ameliaOutboundMessage) {

    if (ameliaOutboundMessage.attributes != null && ameliaOutboundMessage.attributes.formInputData != null

      // This message contains form data that we need to handle

      displayFormMessage(ameliaOutboutMessage, new MyFormFinishedListener() {

          // This is called by the client when the form is submitted, the formInputData parameter represen
          // the same formInputData as originally received but some FormInputFieldOption has likely
          // been modified with a call to FormInputFieldOption#setSelected(boolean)
          // It is safe (and recommended) to use the same instance of FormInputData as received by the sdk
          // The utterance string is the FormInputFieldOption#getValue() of the selected option OR
          // the FormInputField#getName() of the selected field in case it contains zero options
          public void onFormSubmitted(FormInputData formInputData, String utterance) {
            ameliaChat.submitForm(formInputData, utterance);
          }

      });
    } else {

      // Handle as a normal message
      displayMessage(ameliaOutboundMessage);
  }
}
```

See the javadoc for `FormInputData`, `FormInputField` and `FormInputFieldOption` for details about the available properties.

In general, a `FormInputData` contains an array with a small number of `FormInputFields`. A `FormInputField` can contain zero or more `FormInputFieldOptions`. A field without options is to be regarded as a simple button while a field with options should should be presented as choice of several options. The details are intentionally loosely specified and requires the client and the Amelia instance to be configured so as to achieve the desired results.

Integration message

Integration messages are reported on the IConversationListener as an outboundIntegrationMessage(...). The `AmeliaOutboundMessage` received will have a non-null `integrationMessageData` on its `AmeliaOutboundMessageAttributes`

This data is entirely freeform and it is up to to client to interpret and act on the data.

The general flow is usually that the client is expected to present a user interface and then subsequently start a named process by calling `ameliaChat.runAction(...)`

Data Masking

When Amelia asks a question that requires responses to be secured, e.g. password, credit card number, the user input needs to be masked. The SDK will fire a change event to notify the developer that the secure input state has changed. Here is an example:

```
    builder.addConversationListener(new BaseConversationListener() {
        @Override
        public void onSecureInputEnabledChanged(boolean enabled){
            if(inputField!=null) {
                if(enabled) {
                    inputField.setInputType(InputType.TYPE_CLASS_TEXT |
                            InputType.TYPE_TEXT_VARIATION_PASSWORD);
                }else{
                    inputField.setInputType(InputType.TYPE_CLASS_TEXT);
                }
            }
        }
});
```

Only when the secure input enabled state is changed, this event will be fired. Otherwise it indicates the secure input state stays the same. The initial state at new conversation is always false.

*Note: sdk-sources.jar has been provided for convenience for development. It contains all the necessary API code and documentations.