Contents

Amelia SDK 3.1.3

Prerequisites

This guide assumes Android Studio 3.0.1 and Gradle 4.2 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-3.1.3.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:3.1.3@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("https://your-amelia-host.xyz.com")
    .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<BaseDomain> domains) {
        ameliaChat.selectDomain(domains.get(0));//or present domain selection UI for user to select one
    }

    @Override
    public void onConversationStart() {
        Toast.makeText(context, "Conversation started!", Toast.LENGTH_SHORT).show();
    }

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

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

    @Override
    public void onConversationFail(IAmeliaError error) {
        Toast.makeText(context, error.getMessage(), 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 outboundTextMessage(IAmeliaOutboundMessage body) {
        Toast.makeText(context,
                body.getFromUserDisplayName() + " says: " + body.getMessageText(),
                Toast.LENGTH_SHORT)
                .show();
    }
});
```

Step 4. Create the AmeliaChat instance:

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

Step 4.5

```
ameliaChat.initialize();
```

This will initialize the chat and have sessionInfo returned if in SessionListener. sessionInfo tells if user is currently logged in, if anonymous is allowed, and anonymous user data:

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

         @Override
                public void sessionInitialized(final ISessionInfo sessionInfo) {
                       if(sessionInfo.isAnonymousAllowed()){
                              Log.d(TAG,"anonymous is allowed on this server");
                       }else{
                              Log.d(TAG,"anonymous is not allowed on this server");
                       }
                }

         @Override
         public void onSessionInitFail(AmeliaError error) {
            Log.e(TAG,"Error:"+error.getMessage());
         }
    });
```

Step 5. Start the conversation: start an anonymous chat:

```
    @Override
    public void sessionInitialized(final ISessionInfo sessionInfo) {
           if(sessionInfo.isAnonymousAllowed()){
               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!`

AuthSystem

AuthSystem indicates the type of authentication method. It could either be LDAP internal login or SAML external login. After ameliaChat.startNewConversation() is called, if the server requires logging in, it will return the list of existing AuthSystems in the callback method of ISessionListener:

```
ISessionListener sessionListener = new BaseSessionListener() {
        @Override
        public void onLoginRequired(List<BaseAuthSystem> authSystems) {
        }
}
```

To login against either internal or external login,an AuthSystem needs to be passed to LoginOptions. To choose the correct AuthSystem, you could check the type of AuthSystem by the code below:

```
if(authSystem.getRedirect()){//Amelia server requires a third party to authenticate user
    //This auth system is for external login, and requires app to open the external login link,
    //e.g. SAML
}else{
    //This auth system is internal login, usually against LDAP
     LoginOptions options = new LoginOptions(authSystem, email, password);
     ameliaChat.login(options);
}
```

If you already know the authSystem code set up on the server side, you could also check with:

```
String authCode = authSystem.getCode();
if(authCode!=null&&authCode.equals("<auth code set up on server>"){
}
```

Login methods

Anonymous login is triggered by calling `ameliaChat.newConversation()`

Authenticated login can be triggered via `ameliaChat.stepupLogin()`, which will return `ISessionListener.onLoginRequired(List<E`

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

Anonymous

To check whether anonymous is allowed on the Amelia server, simply call

```
    ameliaChat.initialize();
```

before ameliaChat.newConversation() or ameliaChat.stepupLogin is called. initialize() will return asynchronously the sessionInfo in the SessionListener, which tells if anonymous is allowed:

```
new BaseSessionListener() {

        @Override
               public void sessionInitialized(final ISessionInfo sessionInfo) {
                   if(sessionInfo.isAnonymousAllowed()){
                       ameliaChat.startNewConversation()//start an anonymous chat if anonymous is allowed
                   }else{
                       ameliaChat.stepupLogin()//start an authenticated chat if anonymous is not allowed
                   }
               }

        @Override
        public void onSessionInitFail(AmeliaError error) {
           Log.e(TAG,"Error:"+error.getMessage());
        }
    }
```

When both anonymous and authenticated login are supported on the server, call stepupLogin to force authenticated login:

```
    ameliaChat.stepupLogin();
```

Internal login

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

```
LoginOptions options = new LoginOptions(authSystem, email, password);
 ameliaChat.login(options);
```

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`. You can use a webview or a CustomChromTab to capture the cookie. For a webview, the cookie will be automatically saved and once login is complete, just close the webview and call:

```
 LoginOptions loginOptions = new LoginOptions(authSystem, getActivity().getApplicationContext());
 ameliaChat.login(loginOptions);
```

With a chrometab, the cookie can be captured in the redirect url, with a proper host and scheme setup, on the server side and in AndroidManifest. For example, if host="login" and scheme is "ameliaclientapp", then url will be: ameliaclientappv3://login?cookie=[amelia-session-cookie]

To login with a session cookie:

```
 LoginOptions loginOptions = new LoginOptions(authSystem, sessionCookie)
 ameliaChat.login(loginOptions);
```

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

Initial Attributes and initial BPN variables

Initial attributes can be passed into `startNewConversation(Map<String,String> initialAttributes, Map<String,String> initialBpnVariables)` to start an anonymous conversation. For authenticated login, pass initial attributes into the LoginOptions:

```
   LoginOptions options = new LoginOptions(authSystem, email, password);
   Map<String, String> attributes = new HashMap<>();
   attributes.put("age", "25");
   options.setInitialAttributes(attributes);
   ameliaChat.login(options);
```

To include initialBpnVariables to kick off a bpn on conversation start, pass the variables into the second parameter of startNewConversation to start anonymous conversation or, for authenticated login:

```
LoginOptions options = new LoginOptions(authSystem, email, password);
   Map<String, String> bpnVariables = new HashMap<>();
   bpnVariables.put("processName", "attr");
   options.setInitialBpnVariables(bpnVariables);
   ameliaChat.login(options);
```

Note that initialAttributes and initialBpnVariables are all optional. If only passing one of them, just pass null for the other:

```
startNewConversation(attributeMap,null);
```

Domains

List of Domains are automatically returned in the callback method of SessionListener after a successfully login or after IAmeliaChat.startNewConversation() is called (to start an anonymous chat). Provided there were no errors during login or initializing the session;

```
ISessionListener sessionListener = new BaseSessionListener() {
        @Override
        public void onDomainSelectionRequired(List<BaseDomain> domains) {
            ameliaChat.selectDomain(domain.get(0));
        }
}
```

A session listener can be added by calling

```
ameliaChat.addSessionListener(ISessionListener)
```

or adding it during the creation of AmeliaChat:

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

When domain selection is automatic, and there is only one domain, then it will be selected. If there are multiple domains, domainfail will be sent. When manual is chosen, the list of domains will be returned to onDomainSlectionRequired. Note: When not using the session listener anymore, you should remove it from ameliaChat:

```
ameliaChat.removeSessionListener(sessionListener);
```

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();
```

Ignore Certificate Errors

This API is added for testing with servers that have invalid SSL certificates or self-signed SSL certificates The default value for 'ignoreCertificvate' is false. In production environment this API should not be used.

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

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

Speech recognition (STT)

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 `IDownloadMessage` contains necessary information to download associated files. Please note: Sample code to listen to and act on `outboundMmoDownloadMessage(...)`:

```
builder.addConversationListener(new BaseConversationListener() {
    @Override
    public void outboundMmoDownloadMessage(IAmeliaOutboundMessage ameliaOutboundMessage) {
        IDownloadMessage downloadMessage = ameliaOutboundMessage.getDownloadMessage();
        if (downloadMessage.getError() != null) {
            downloadMessage.download(downloadListener, context);
        }
    }
});
```

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

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

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

    @Override
    public void onDownloadSuccess(IDownloadedMmo mmo) {
        Toast.makeText(MainActivity.this, "Downloaded: " + mmo.getUri(), 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(IAmeliaOutboundMessage ameliaOutboundMessage) {
        Toast.makeText(MainActivity.this, "Upload requested", Toast.LENGTH_SHORT).show();
        openFileChooser(ameliaOutboundMessage.getFromUserDisplayName(),
                                            ameliaOutboundMessage.getUploadMessage().fileType);
    }

    @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 `IAmeliaOutboundMessageAttributes` on a received `IAmeliaOutboundMessage` 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 outboundFormInputMessage(IAmeliaOutboundMessage ameliaOutboundMessage) {

    if (ameliaOutboundMessage.getAttributes() != null &&
                                ameliaOutboundMessage.getAttributes().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 represents 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 `IAmeliaOutboundMessage` received will have a non-null `integrationMessageData` on its `IAmeliaOutboundMessageAttributes`

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.