cloudonix
MAXIMUM COMMUNICATION

Cloudonix Mobile Application - Developer Guide

Introduction

This is the mobile application developer guide for the Cloudonix Mobile SDK release 5.0. It will help you integrate the Cloudonix Mobile SDK into your Android or iOS mobile application.

Getting Started

The Cloudonix Mobile SDK is distributed with a sample dialer application to help you get started. The sample dialer application demonstrate how to use the SDK to connect to a SIP service and allows the user to call through the SIP service using a dial pad or a contact list.

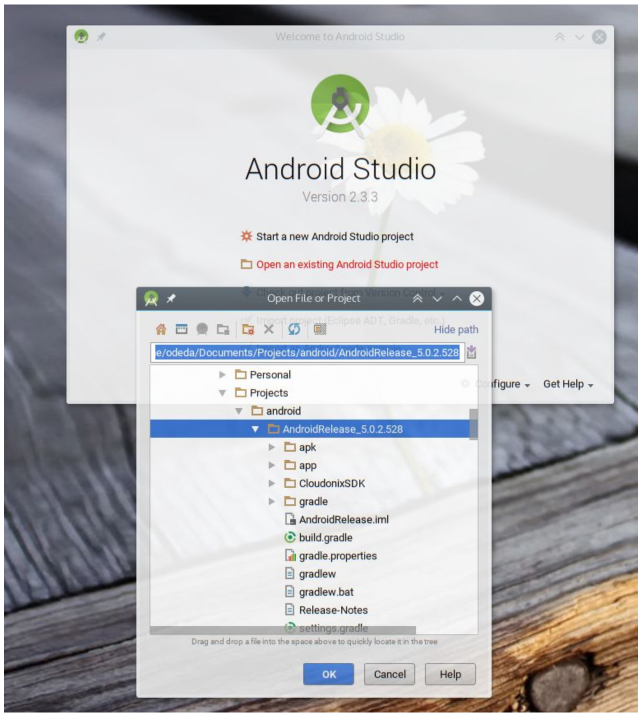
The following sections cover setting up the Cloudonix Mobile SDK sample application and running it.

Android

Please note that Cloudonix Mobile SDK release 5.0 is supported on Android Studio up to version 2.3.3.

Project Setup

Download the release package and extract it in your projects folder. Then use the Android Studio to open the project:


Welcome to Android Studio
Android Studio
Version 2.3.3
Start a new Android Studio project
Open an existing Android Studio project
Open File or Project
Hide path
Configure
Get Help
OK
Cancel
Help

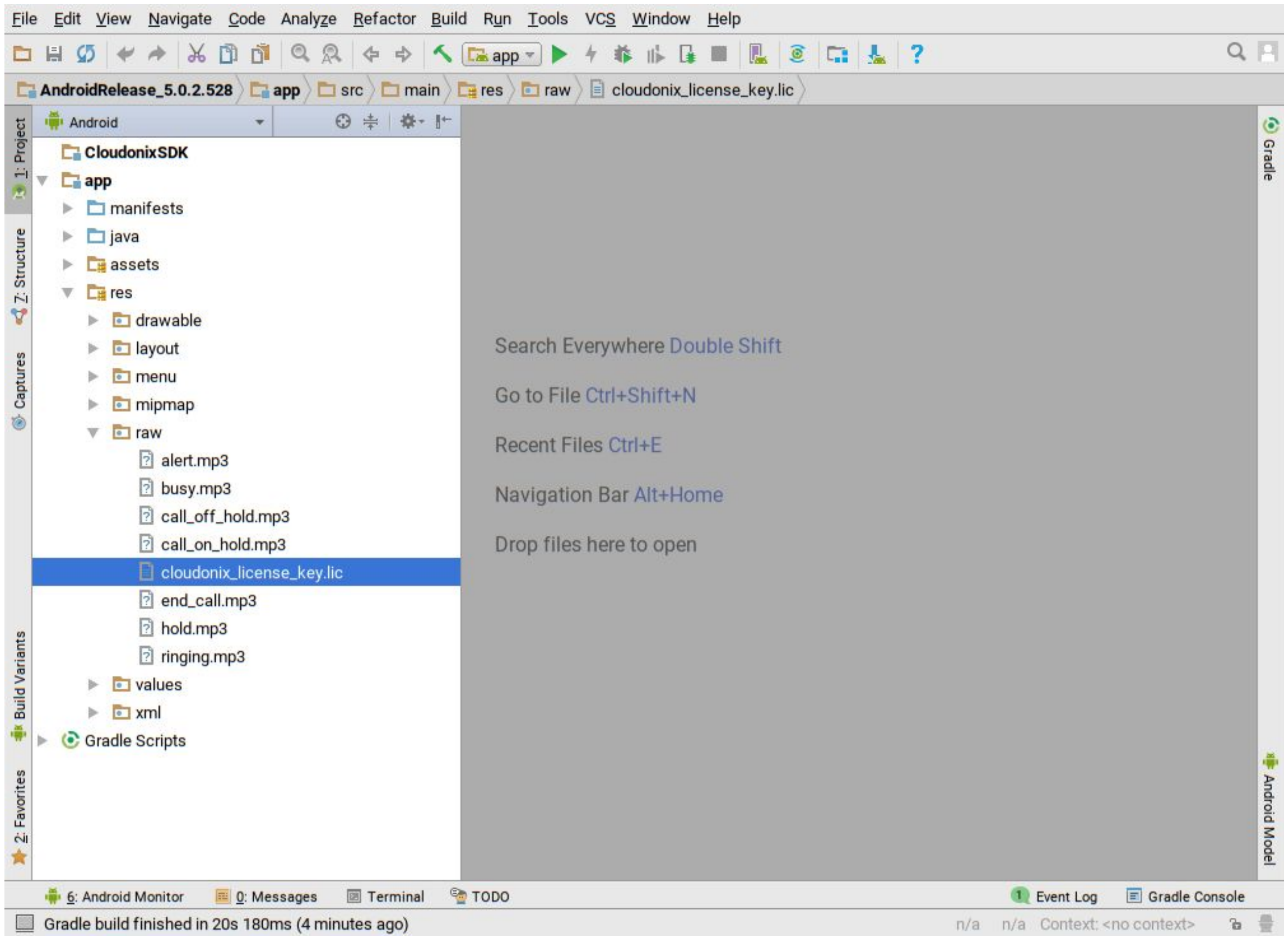
e/odeda/Documents/Projects/android/AndroidRelease_5.0.2.528

```
Personal
Projects
  android
    AndroidRelease_5.0.2.528
      apk
      app
      CloudonixSDK
      gradle
      AndroidRelease.iml
      build.gradle
      gradle.properties
      gradlew
      gradlew.bat
      Release-Notes
      settings.gradle
```

Drag and drop a file into the space above to quickly locate it in the tree

Deploying License Key

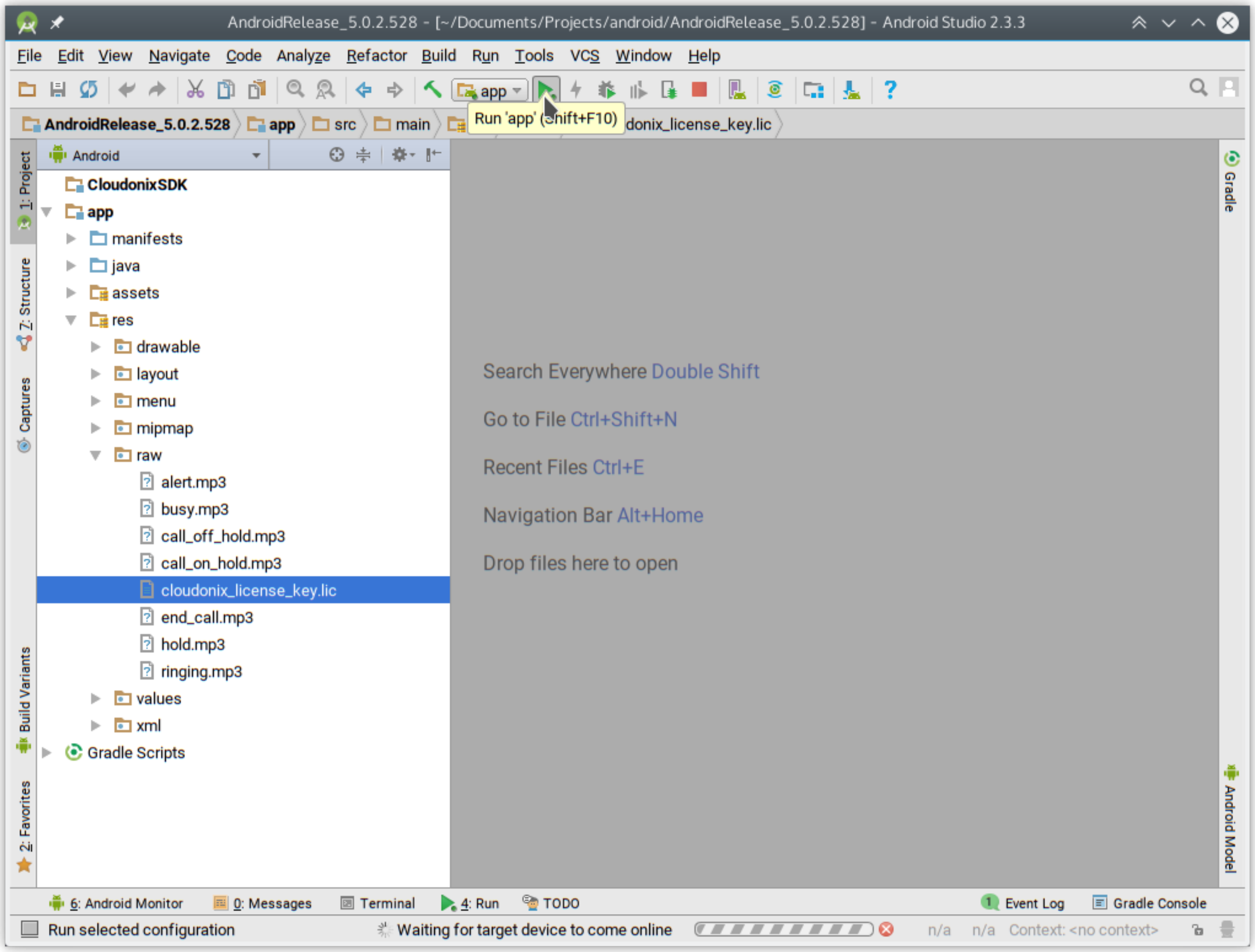
After the project loads, navigate to `app` > `res` > `raw` , and replace the empty place-holder file `cloudonix_license_key.lic` with the license file you've received from Cloudonix.



The file name should either be kept the same as in the sample project, or otherwise you can change the name of the file loaded by the sample application by editing the `loadLicenseKey()` method in the `net.Cloudonix.iotech.cloudonixsdk.cloudonixdialer.utils.VoIPClient` class.

Build and Run Project

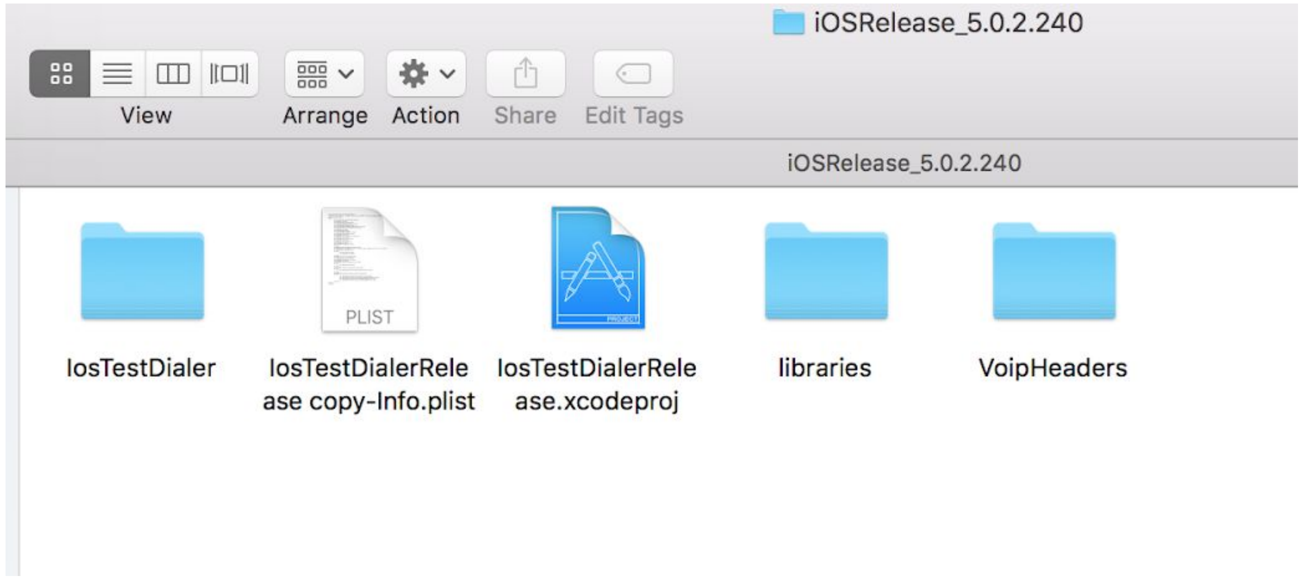
At this point you should be able to click the “Run” button to start the sample dialer application on a connected device or an emulated device.


AndroidRelease_5.0.2.528 - [~/Documents/Projects/android/AndroidRelease_5.0.2.528] - Android Studio 2.3.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
Run 'app' (Shift+F10)
AndroidRelease_5.0.2.528 app src main donix_license_key.lic
Android
CloudonixSDK
app
manifests
java
assets
res
drawable
layout
menu
mipmap
raw
alert.mp3
busy.mp3
call_off_hold.mp3
call_on_hold.mp3
cloudonix_license_key.lic
end_call.mp3
hold.mp3
ringing.mp3
values
xml
Gradle Scripts
Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open
6: Android Monitor
0: Messages
Terminal
4: Run
TODO
Event Log
Gradle Console
Run selected configuration
Waiting for target device to come online
n/a n/a Context: <no context>

iOS

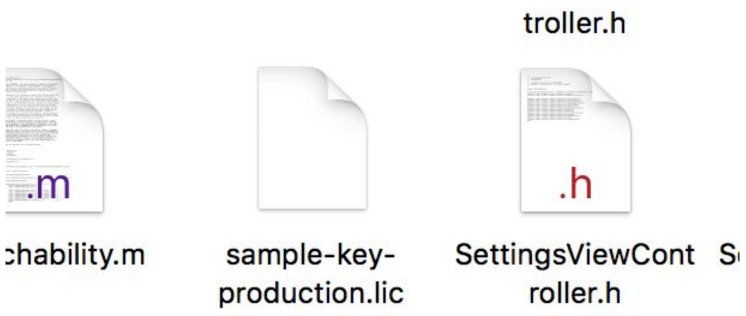
Project Setup

Download the release package and extract it in your projects folder.



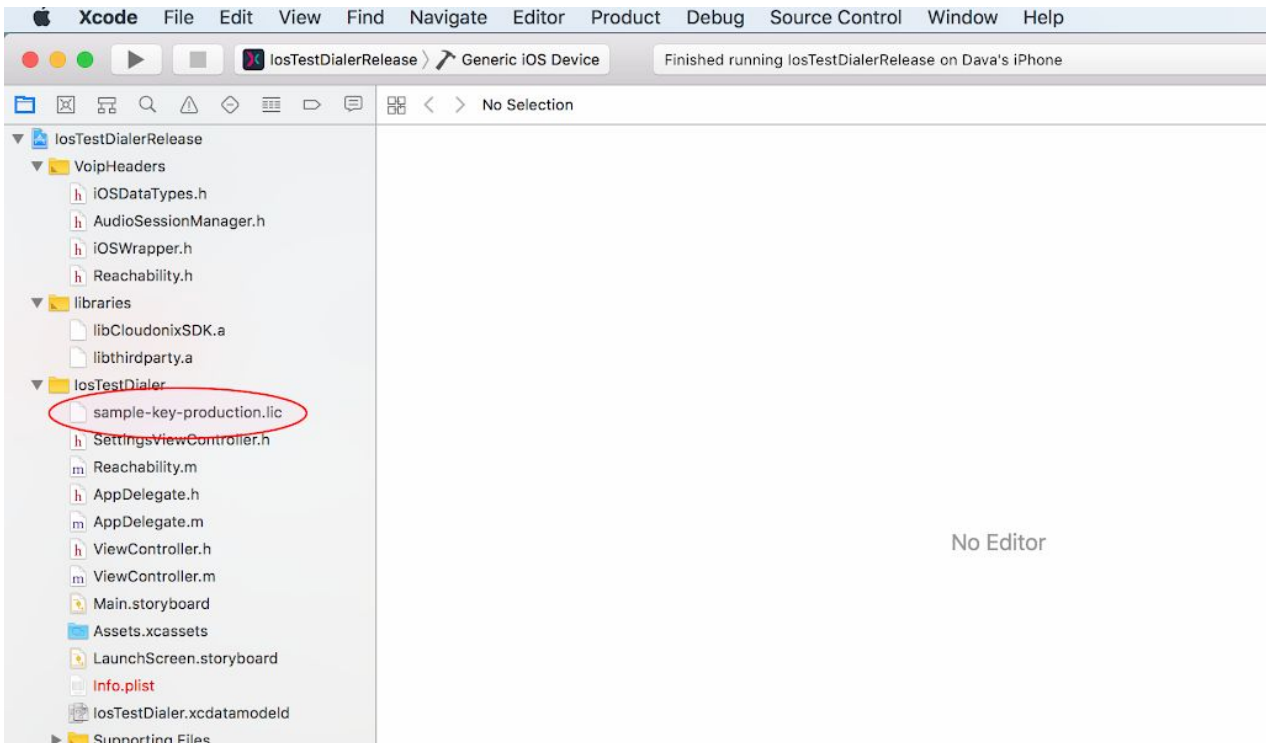
Deploying License Key

You should have an `IosTestDialer` folder in the project directory, into which the license key file should be copied, and renamed to `sample-key-production.lic`



Build and Run Project

Open the sample application project file `IosTestDialerRelease.xcodeproj` to open the project in Xcode, then use the “Build and Run” command from Xcode:



Application Workflows

Initialization

In order to use the Cloudonix Mobile SDK in your application, it must first be initialized with the license key (which you receive from Cloudonix, either a trial or a production license key). The application would load the license key into the Cloudonix Mobile SDK in order to retrieve an instance to the Cloudonix Mobile SDK control object (commonly called “instance” in this guide). This process will verify the license, configure the SDK for the device by utilizing Cloudonix online device configuration service and will eventually issue the `onLicense` event, which can be monitored by the application to check if the licensing process completed successfully. If the licensing process failed - for example if the license key is corrupt or has expired - then the SDK will not be operational.

After initializing the SDK, it needs to be configured with configuration parameters such as allowed codecs and transports, in addition to setting the SIP account details that will be used for calling and receiving calls.

Android

After receiving the SDK client instance, the application should perform these additional steps:

1. Call `addEventsListener()` to be able to receive events from the SDK using callbacks on a `IVoIPObserver` instance.
2. Call `initPreferences()` which is needed for the Android Cloudonix Mobile SDK service to be able to handle incoming calls to the application even when it isn't in the foreground.
3. Use `setConfig()` to set up the SDK.
4. Call `checkBinder()` to see if the SDK service is still running and can be reused. If the service is running, this call will attach to the existing service.
5. If `checkBinder()` returned `false`, the application needs to call `bind()` to create and connect a new instance of the SDK service. After that process is complete, the SDK will fire the `onLicense()` event with `LicensingState` set to `LICENSING_SUCCESS` to let the application know that the service is ready and licensed correctly.
6. At this point the application can call `setConfig()` to override any default configuration parameters of the SDK then call `setConfiguration()` with the SIP account registration information.
7. The `setConfiguration()` call will instruct the SDK to setup the SIP stack and when process is complete, the SDK will fire the `onSipStarted()` event.
8. Once `onSipStarted()` is called, the application can start communication flows such as registering and dialing.

Android Example Initialization

```
public class VoIPClient implements IVoIPObserver {
  CloudonixSDKClient instance;

  Private initSDK(Context ctx) {
    InputStream input = ctx.getResources().openRawResource(R.raw.license);
    try {
      String lic = new String(CryptUtils.convertStreamToByteArray(input));
      instance = CloudonixSDKClient.getInstance(lic);
      instance.addEventsListener(this);
      instance.initPreferences(ctx);
      if (instance.checkBinder())
        return;
      instance.bind(); // will cause onSipStarted to be called
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  @Override
  public void onLicense(LicensingState state, String description) {
    // handle licensing problems
    instance.setConfig(ConfigurationKey.USER_AGENT, "MyApp/1.0");
    instance.setConfiguration(new RegistrationData() {{
      setServerUrl("sip.server.net");
      setDomain("my.service.com");
      …
    }});
  }

  @Override
  public void onSipStarted() {
  }
}
```

Registration and Registration-Free Setup

The Cloudonix Mobile SDK supports both classic SIP accounts using periodic REGISTER messages to maintain “connection” to the server, as well as Cloudonix Registration-Free mode.

Classic mode (“registration”)

In a classic SIP registration use case, the application should:

1. Make sure the RegistrationData contains the correct username and password during the configuration step.
2. In order to receive calls, call `registerAccount()` after the Cloudonix Mobile SDK notifies the application that the SIP stack has started through the `onSipStarted()` event.

After calling `registerAccount()` the Cloudonix Mobile SDK will fire the `onRegisterState()` event whenever the registration state changes. The application can also call `isRegistered()` to check the registration status.

Please note that for the `registerAccount()` call to succeed, the user authorization credentials should have already been set using the `setConfiguration()` call.

Regardless if the application has called `registerAccount()` or not, after the SIP stack was started, the application may call `dial()` or `dialWithHeaders()` to start a call using the configured user authorization credentials.

Android Example Registration

```
@Override
public void onLicense(LicensingState state, String description) {
  // handle licensing problems
  instance.setConfig(ConfigurationKey.USER_AGENT, "MyApp/1.0");
  instance.setConfiguration(new RegistrationData() {{
    setServerUrl("sip.server.net");
    setDomain("my.service.com");
    setUsername("972547340014");
    setPassword("secret");
    …
  }});
}
```

```
@Override
public void onSipStarted() {
    instance.registerAccount();
}

@Override
public void onRegisterState(RegisterState result, int expiry) {
  switch(result) {
  case REGISTRATION_SUCCESS: logger.d(TAG, "registered"); break;
  case REGISTRATION_ERROR_CREDENTIALS: logger.d(TAG, "auth error"); break;
  case REGISTRATION_UNREGISTERED: logger.d(TAG, "No longer registered"); break;
  }
}
```

Registration-Free mode

In a Registration-Free use case, there is no need for the application to perform SIP registration as call reception is handled by the application backend using push notifications, and there is also no need to set user authorization credentials as the application will be authorizing each call separately using a Cloudonix session token.

At any point after the Cloudonix Mobile SDK fires the `onSipStarted()` event. The application can dial or receive calls, using the `dialRegistrationFree(number, token)` call.

Dialing In Classic Registration Mode

After the application completes setting up the SDK and configuring the SIP account details, the application may use the `dial()` command to start a SIP session.

The dial command will start the calling process and will cause the Cloudonix SDK to issue `onCallState()` events for each stage in the call progress.

Alternatively, `dialWithHeaders()` may be used to add custom SIP headers to the SIP call.

Android Example Dial in Registration Mode

```
public void dial(String number) {
  dial(number);
}

@Override
public void onCallState(String callId, CallState state, String contactUrl) {
  Switch (callState) {
  …
  }
}
```

Dialing in Registration-Free Mode

After the application completes setting up the SDK and configuring the SIP account details (usually just the SIP server address and domain name), the application may start a Registration-Free dial by obtaining a session token from application backend (please refer to the Registration-Free specification for more details) and then use the `dialRegistrationFree(number, token)` command to start a SIP session.

It is also possible to set up the SIP stack only right before making a call (or receiving a call). This is useful in case the application manages multiple subscriber accounts and knows which account to use for a call only at the last second. In such a case it is important to wait for the `onSipStarted()` event before issuing the dial call, for example:

Android Example Dial in Registration-Free

```
private void dial(String msisdn, String token, RegistrationData regdata) {
  // listen Cloudonix SDK events
  instance.addEventsListener(new IVoIPObserver() {
    @Override
    public void onSipStarted() {
      instance.dialRegistrationFree(msisdn, token);
    }
  });

  instance.setConfiguration(regdata);
}
```

Call Reception in Class Registration Mode

When a call is received, the `onCallState()` event will be invoked with the `CallState` set to `CALL_STATE_INCOMING`. When the application handles the event, it can call either `answer(callId)` or `reject(callId)` as required.

Call Reception in Registration-Free Mode

When a push notification for an incoming call is received, containing a Registration-Free call token, the application should make sure that `onSipStarted()` has already been received, and then use the `dialRegistrationFree(msisdn, token)` call to accept the call.

Please note that the phone number provided in the first parameter can be of any value (the token is the only value that determines the success of receiving the call), but its value must never be set to `null`.

Application Shutdown

It is important to note that when the application is moved to the background, the SDK will still maintain context to allow the application to receive calls. If the application needs to shut down completely and not receive any calls, it should call `shutdown()`, after which the SDK is completely shut down and in order to start it again it needs to be reinitialized.

Reference

SDK Integration Notes

Android

iOS - Xcode

The Cloudonix Mobile SDK uses the following libraries from the iOS SDK:

- `VideoToolbox.framework`
- `GLKit.framework`
- `libstdc++.tbd`
- `libicucore.tbd`

While these libraries are not automatically linked in to the project, they are available through the iOS SDK.

To link the project with these libraries (in case the application isn't already using them):

1. In the project's setting, change to the "Build Phases" view.
2. In the "Link Binary With Libraries" section, click the plus sign to add a new library.
3. Add each of the libraries listed above.