The Cloud Alert App Embeddability for Android

Developers Guide

Table of Contents

Introduction

Cloud Alert is a comprehensive solution for the alerting and notification requirements of banks and other financial enterprises. It is a combination of mobile app, cloud messaging service and messaging gateway that enables an organization to instantly communicate with their customers.

The Cloud Alert App is a mobile app that serves as the user interface for the Cloud Alert service. The app is installed on the cardholder's handset, and displays information about his transactions, banking information and other notifications that the bank sends to him.

While the Cloud Alert App can be installed and used as a standalone app, it can also be embedded into the bank's existing mobile app. Doing so allows the bank to roll out the service as an update to their app, without having to request users to install another app.

This document details how to embed the Android edition of the app. For other platforms, contact support at *support@cloudalert.cloud*.

Requirements

Before you begin, the following are the requirements:

- Software development tools for Android (Android Studio)
- The AAR file provided by Cloud Alert
- Your organization's Subscriber ID at Cloud Alert

Software development tools

This document assumes the use of Google's Android Studio, which is freely downloadable at

`https://developer.android.com/studio`

This document also assumes knowledge of the Java programming language and the essentials of developing apps for Android in Java.

The AAR and the instructions in this document have been tested with Android Studio v3.3.2 on Linux.

The Cloud Alert AAR File

An Android Archive (commonly known by its suffix .aar) is a software library that bundles an entire app into a library. An AAR can be embedded into any Android app to provide additional functionality to that app.

The Cloud Alert AAR is provided by Cloud Alert to your organization. This file contains the complete functionality of the Cloud Alert App, including the activities for viewing messages, settings and initial setup, as well as the background service that connects to the server.

Since the file is typically rebranded to your organization's brand, logo and colors, your organization will have to provide these resources to Cloud Alert to receive your customized AAR. Details on the rebranding process are present in the Cloud Alert App Rebranding Guide.

For purposes of testing, a generic AAR file is also available. While this AAR contains the Cloud Alert brand and logo, it is technically identical to any provided AAR, and so can be used for development until a rebranded AAR is available.

Note that integrating with the AAR does not add another icon to the handset's applications. Launching your app is the only way to enter into the Cloud Alert app. While it is possible to include a separate icon just for the Cloud Alert functionality, it is beyond the scope of this guide.

Note that while the AAR is significantly built in Java, it includes portions that utilize native code for performance reasons. Since ARM32, ARM64, x86 and x64 are supported by the native portions, all handsets in the market today are supported. Support for niche processors is available on special request.

Subscriber ID

Your organization is allocated a Subscriber ID, which is a 64-bit number uniquely identifying your organization. This ID needs to be passed to the Cloud Alert app on launch. The value of 0 can optionally be passed for testing purposes, but a valid value is recommended for production use.

Integrating the AAR

Integrating the AAR into your app includes the following steps:

- Choosing key parameters
- Choosing the point of integration
- Understanding the modes of operation
- Importing the AAR into your project
- Invoking the AAR

Choosing key parameters

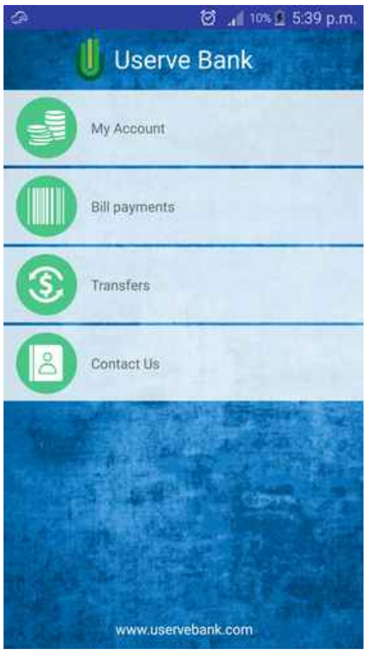
One of the key parameters to decide on is your app's Minimum SDK Version (field minSdkVersion), which is typically set in your app's Android Manifest, or your build.grade file. This value in your app should be equal to higher than the version supported by the AAR.

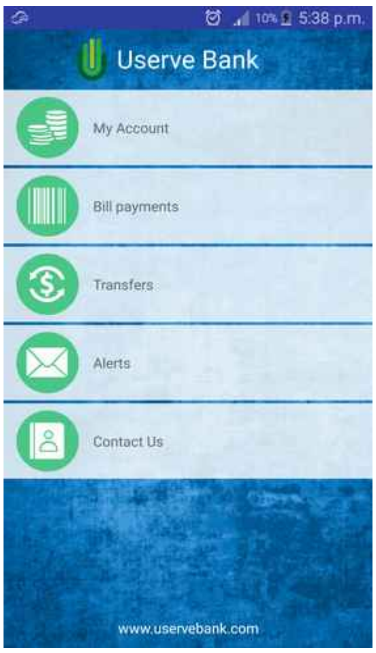
The current minimum SDK version of the AAR is 16, which means Android versions 4.1 (Jelly Bean) and higher are supported, ie, over 95% of the Android user base.

Choosing the point of integration

Typically, your app will include a launcher-style main menu, with functionality that the user can select, for example:

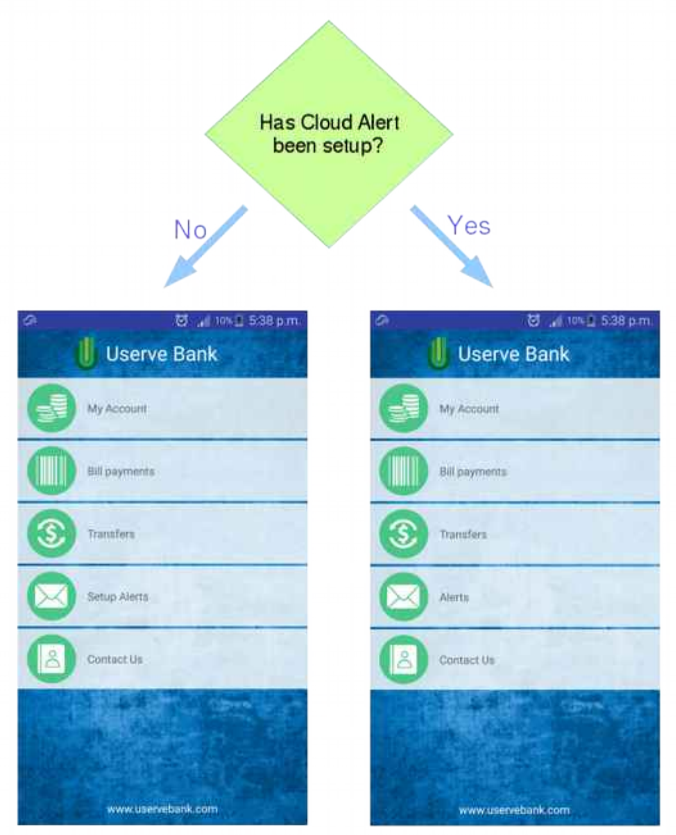
Cloud Alert can be added as one of the items on the menu, like so:





Before the app is used for the first time, it needs to be setup on the handset. It is recommended that your app determine whether Cloud Alert has been setup, and display a different text in your launcher if not setup, as displayed in the figure.

Thus, when the user clicks on the button for the first time, the app performs its setup. On subsequent clicks, the app is launched to display messages to the user.

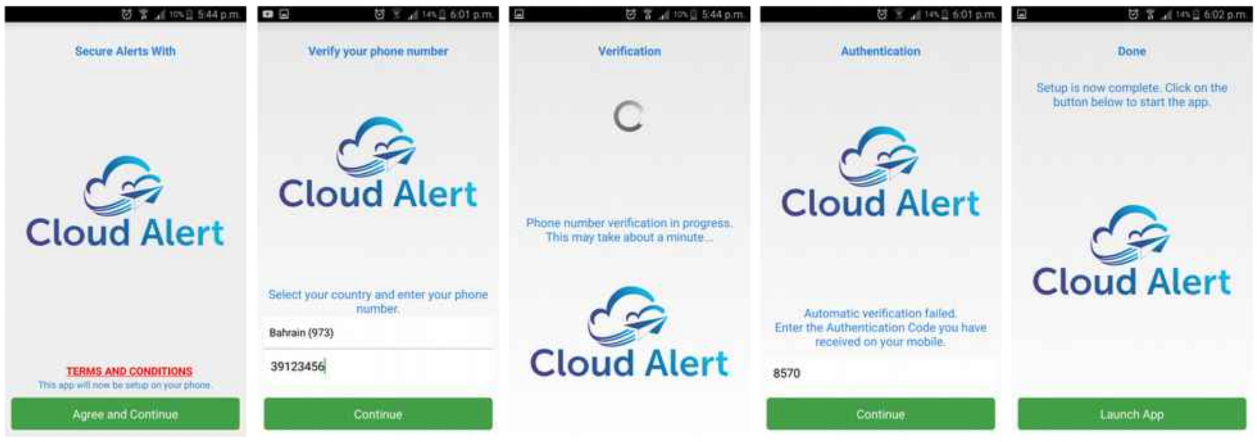


Understanding the modes of operation

Setup of the app

When your app invokes the setup of the Cloud Alert app, a wizard of activities is displayed to the user which will guide the user through the setup process. The setup typically involves:

- A welcome page with terms and conditions
- A page that requests the user's phone number
- A page that signs up with the Cloud Alert service, receives the authentication sms and receives the security certificate.
- A completion page.


Secure Alerts With
Cloud Alert
TERMS AND CONDITIONS
This app will now be setup on your phone.
Agree and Continue

Verify your phone number
Cloud Alert
Select your country and enter your phone number.
Bahrain (973)
39123456
Continue

Verification
Phone number verification in progress. This may take about a minute...
Cloud Alert

Authentication
Cloud Alert
Automatic verification failed. Enter the Authentication Code you have received on your mobile.
8570
Continue

Done
Setup is now complete. Click on the button below to start the app.
Cloud Alert
Launch App

Should your app already know the user's phone number, it should pass it to the Cloud Alert app. Doing so will bypass the page that requests the user for his phone number. The phone number must match the number of the user's SIM card, for authentication of the SIM through SMS.

Note that if the handset does not receive the sms, the user is prompted for the authentication code. This allows a user to install the app through other means of authentication, or on a device other than the one that receives the sms. This feature can also be used if the sms delivery to the handset is delayed.

Once the setup has exited, your activity that launched Cloud Alert is brought to the foreground. Your activity must now determine whether the setup was successful, and correspondingly update the text of your link.

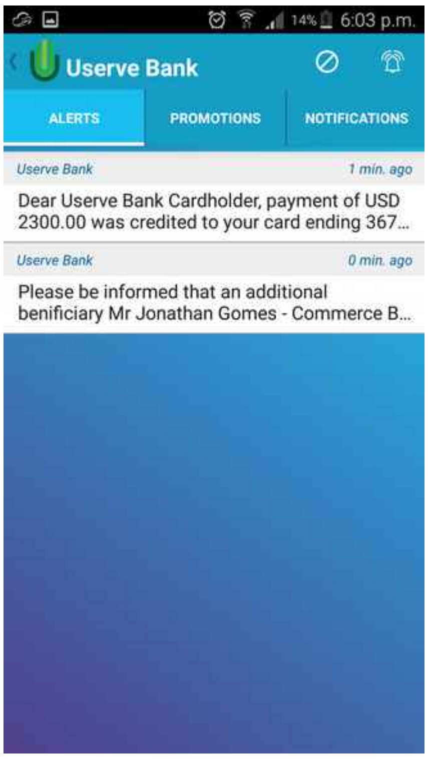
Launching the app

Your user can now launch the app by clicking on the menu item.

This activity displays the messages sent by your organization.

The activity contains tabs at the top for each of the message categories. The user can click on the tabs or swipe to switch between categories.

Note that the functionality of the app is altered if your application does not pass the Subscriber ID to the Cloud Alert App. In that case, an additional activity is displayed where the user can choose the sender.



Userve Bank 1 min. ago

Dear Userve Bank Cardholder, payment of USD 2300.00 was credited to your card ending 367...

Userve Bank 0 min. ago

Please be informed that an additional benificiary Mr Jonathan Gomes - Commerce B...

Importing the AAR into your project

To get started with the integration:

- Launch Android Studio
- Select **File** -> **New Module**
- Select **Import .JAR/.AAR Package** then click **Next**
- Enter the location of the AAR then click **Finish**.
- Next, in the **app** folder, add the below lines to your **build.gradle**, in the respecitve sections:
    - In the 'android' section add the lines marked in red into the respective sections:

```
android {
        packagingOptions {
            pickFirst 'lib/armeabi/libscrypt.so'
            pickFirst 'lib/armeabi-v7a/libscrypt.so'
        }
}
```

    - In the 'dependencies' section add the below lines:
      (rename 'cloudalertembedfile' to the actual filename of the .aar file, without the extension)

```
dependencies {
        implementation project(":cloudalertembedfile")
}
```

    - If you haven't included support for Google APIs, then in the 'dependencies' section add the below lines:

```
dependencies {
        implementation 'com.google.android.gms:play-services-auth:16.0.1'
}
```

- Click on **Sync Now**
- Next, ensure that the below entry has been automatically added to your settings.gradle :

```
include ':app', ':cloudalertembedfile'
```

- Select **File** -> **Invalidate Caches**/**Restart**, then **Invalidate and Restart**.

The AAR should now be present in your project.

Notes

1. The above examples assume that the module for your mobile app will be named 'app'.
2. The entry is to be added to the build.gradle for your app's module, not the project's build.gradle.

Invoking the AAR

Your app needs to call functions present in the AAR to utilize its functionality. This code is typically added to the menu/launcher activity from which Cloud Alert is invoked.

First, the packages need to be imported into your Java files. Ensure that you import only the version of the API that you require. For example,

```
import cloud.cloudalert.app.embedapi.v1.*;
```
imports version 1 of the API.

Mixing imports of different versions will lead to unpredictable results.

Below is sample code to be executed on the click of the menu option:

```
if( ! API.isAppSetupComplete())
{
   SetupInfo setupinfo = new SetupInfo();
   setupinfo.HandsetPhoneNumber = ....
   API.startSetup(this, setupinfo);
}
else
{
   long subscriberid=....
   API.launchApp(this, subscriberid);
}
```
The function **isAppSetupComplete()** determines which of the operations is to be invoked.

The **SetupInfo** object provides information to the setup process. Its **HandsetPhoneNumber** field can be set to the user's phone number (the phone number should include the country-code prefix without the leading zeros). If the phone number is not known, this value can be set to null or a blank string - the user will then be requested for his phone number during the setup process.

The **subscriberid** is provided by Cloud Alert. Should you not have it, pass 0 to test your integration, but keeping in mind that the app will behave differently with it.

The below code can optionally be used to update the caption in the menu:

```
public void onStart()
{
   super.onStart();
   //get the pointer to the button or view
   ...

   if( ! API.isAppSetupComplete())
   {
      button.setText("Setup Alerts");
   }
   else
   {
      button.setText("View Alerts");
   }
}
```

The code is executed in **onStart** so that the button caption will be updated when the user returns after a successful setup.

More information about the above functions and their parameters is present in the Javadoc.

Registering the app hash

During the signup process, an SMS is sent to the handset's SIM with a unique code. For effortless usage, the Cloud Alert app uses Google's SMS Retriever API, which enables it to automatically read this authentication SMS. This mechanism requires the computation of an app hash, which is derived from the app id and the app's public key certificate.

Computation of the app hash is cumbersome and error-prone. To simplify the overall process, the app automatically computes its own hash and send it to Cloud Alert's servers during signup.

Thus, once the embedding of the AAR into your app is complete, you need to perform at least one signup on the new app. Ensure that the app is a Release version, and has been signed with your Production Certificate. During this signup process, the hash is generated and sent to Cloud Alert's servers.

Next, notify Cloud Alert Support that you have successfully perform this initial signup, along with the date-time and the phone number used for signup. Cloud Alert personnel will then register this hash for your app.

For security purposes, unregistered hashes will not be used. The embedded app will still be functional, but the user will have to manually read the code received over SMS and enter it into the Cloud Alert app.

Troubleshooting

Installation displays 'Parse Error:There was a problem parsing the package'

This error might occur during the installation of the app on the mobile device. This indicates that an older version of Android Studio was used to build the apk. Upgrade the Android Studio to v3.3.2 or higher and again generate an apk.

Compilation fails with 'package com.google.gms.auth.api.phone does not exist'

The Google Play API has not been added to your build.gradle.

My app does not read the sms automatically

Ensure that:

- Your handset's 'Google Play Services' is version 11 or newer.
- Your app is using the same app id and certificate that you used to previously register at Cloud Alert's servers.
- The app hash changes when you switch between Debug and Release. You can request Cloud Alert to register both app hashes.

The authentication SMS includes additional characters

The authentication SMS starts with the characters "<#>" and ends with an app hash string of hex values. These are required to tag it as an authentication SMS to Google's SMS Retriever API, and are perfectly normal.