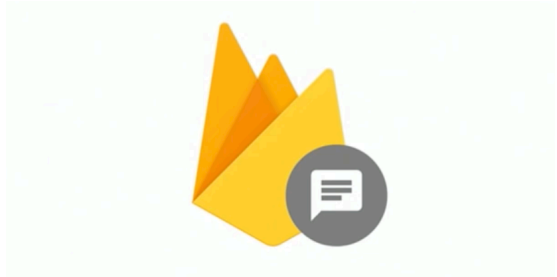
SESSION #8

NOTIFICATIONS
PUBLICATION

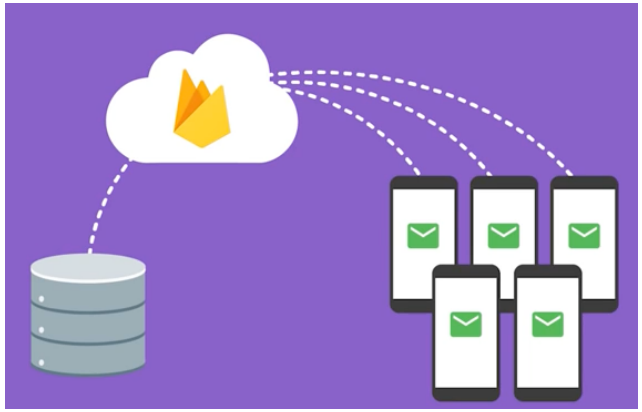
OUTLINE



- Notifications
  - Firebase Cloud Messaging
- Application Publication
  - Signing
- Google Play distribution

NOTIFICATIONS

- User engagement plays a really big role in application development.
- Notifications are those messages that pop in front of a user to grab their attention. And hopefully they make them do something that we want to.
- Notifications are really powerful in getting the users back into our application. Users that opt-in to notifications are twice as likely to come back, as opposed to those that don't.
- If we send too many notifications we might upset the user. We overwhelmed then and we achieve the opposite effect: silence the notification or even uninstall our app.
- We need to know where is the right time to send our notifications.



✧Notifications: https://www.youtube.com/watch?v=KpTSpVh9SfY

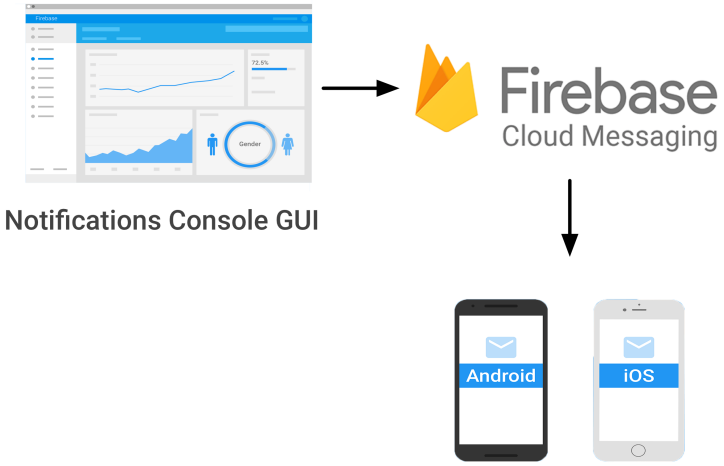
FIREBASE NOTIFICATIONS

- FCM (Firebase Cloud Messaging) is the new version of GCM (Google Cloud Messaging). It has actually inherited all of the infrastructure from GCM, a lot of the APIs and a lot of of the features. If you are integrating messaging in a new app, start with FCM.
- Firebase Notifications is a free service that enables targeted user notifications for mobile app developers. Provides an option for developers and organizations seeking a flexible notification platform that requires minimal coding effort to get started, and a graphical console for sending messages.
- Key capabilities:
  - Notifications analytics: analyse reengagement conversion with built-in notifications analytics collection and funnel analysis.
  - Versatile message targeting: target clients in predefined user segments, custom analytics audiences, clients subscribed to topics, and single devices.
  - Flexible message scheduling: deliver notifications (up to 2kb) immediately, or at a future time in the client's local time.

NOTIFICATIONS CONSOLE

- When your app is in the background on a user's device, notifications are delivered to the system tray. When a user taps on the notification, the app launcher opens your app. If you want, you can also add client message handling to receive notifications in your app when it is already in the foreground on the user's device.
- Implementation path:

1. Set up the FCM SDK. Add one line of code to add the FCM dependency to your app.
2. Send notifications from the Notifications console. Open the Notifications console and start sending notifications to user segments.
3. (Optional) Add message handling. Add message handling logic to your client app in order to receive notifications in your app when it is already in the foreground on the user's device.



NOTIFICATIONS

FCM SDK SETUP

- Prerequisites:
  - Android 2.3 or newer
  - Google Play services 10.0.1 or newer
  - Google Repository from the Android SDK Manager.
  - Android Studio 1.5 or higher
- Add the SDK
  - In your root-level build.gradle:

```
buildscript {
// ...
dependencies {
    // ...
    classpath 'com.google.gms:google-services:3.0.0'
}
}
```

NOTIFICATIONS

FCM SDK SETUP

- Add the SDK
  - In your module Gradle file (usually the app/build.gradle):

```
dependencies {
  // ...
  compile 'com.google.firebase:firebase-core:10.0.1'
  compile 'com.google.firebase:firebase-messaging:10.0.1'
  // Getting a "Could not find" error? Make sure you have
  // the latest Google Repository in the Android SDK manager
}

// ADD THIS AT THE BOTTOM
apply plugin: 'com.google.gms.google-services'
```

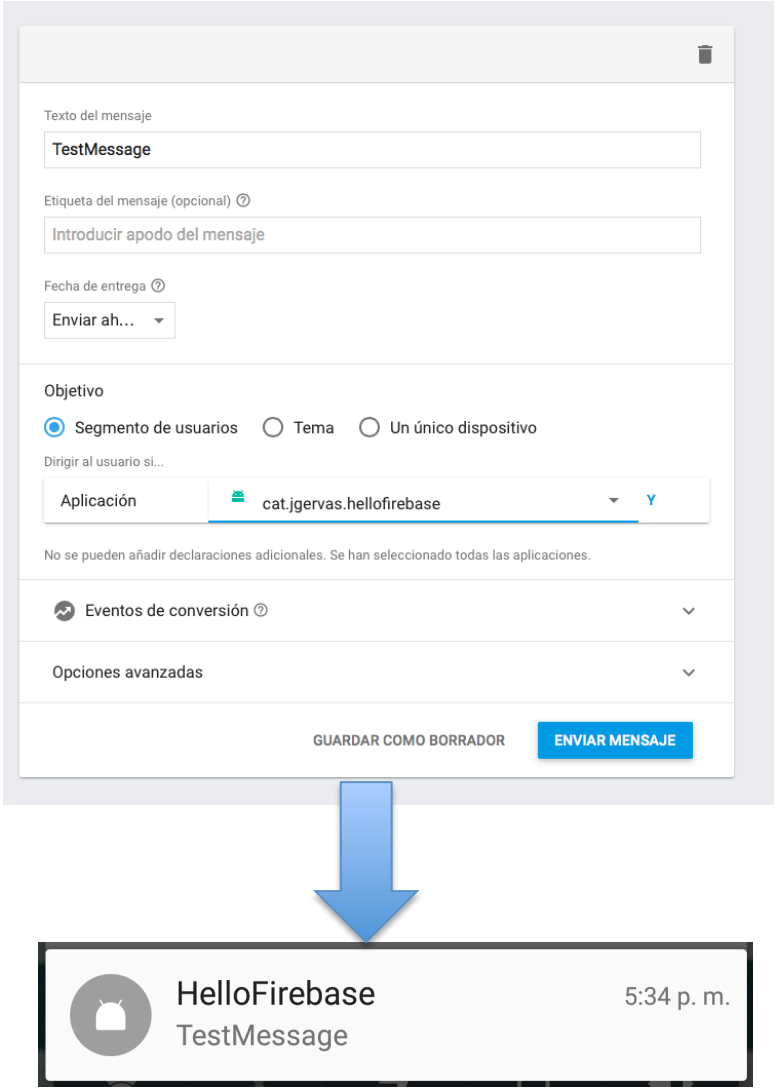
- Add Firebase to your app
  - Create a Firebase project in the Firebase console. Add Firebase to your Android app.
  - Download a google-services.json file. Copy this into your project's module folder, typically app/

NOTIFICATIONS

SENDING A MESSAGE

1. Install and run the app on the target devices.
2. Open the Notifications tab of the Firebase console and select New Message.
3. Enter message text.
4. Select the message target.

✧ After you click Send Message, targeted client devices that have the app in the background receive the notification in the system notifications tray. When a user taps on the notification, the app launcher opens your app.



NOTIFICATIONS

RECEIVE AND HANDLE NOTIFICATIONS

- If you want to receive notifications when your app is in the foreground, you need to add some message handling logic in your client app.
- To receive messages, use a service that extends FirebaseMessagingService. Your service should override the onMessageReceived callback.
- To use FirebaseMessagingService, you need to add the following in your app manifest:

```
<service android:name=".MyFirebaseMessagingService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
```

NOTIFICATIONS

OnMessageReceived

- By overriding the method FirebaseMessagingService.onMessageReceived, you can perform actions based on the received RemoteMessage object and get the message data:

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
    // ...

    // TODO(developer): Handle FCM messages here.
    // Not getting messages here? See why this may be: https://goo.gl/39bRNJ
    Log.d(TAG, "From: " + remoteMessage.getFrom());

    // Check if message contains a data payload.
    if (remoteMessage.getData().size() > 0) {
        Log.d(TAG, "Message data payload: " + remoteMessage.getData());
    }

    // Check if message contains a notification payload.
    if (remoteMessage.getNotification() != null) {
        Log.d(TAG, "Message Notification Body: " + remoteMessage.getNotification().getBody());
    }

    // Also if you intend on generating your own notifications as a result of a received FCM
    // message, here is where that should be initiated. See sendNotification method below.
}
```

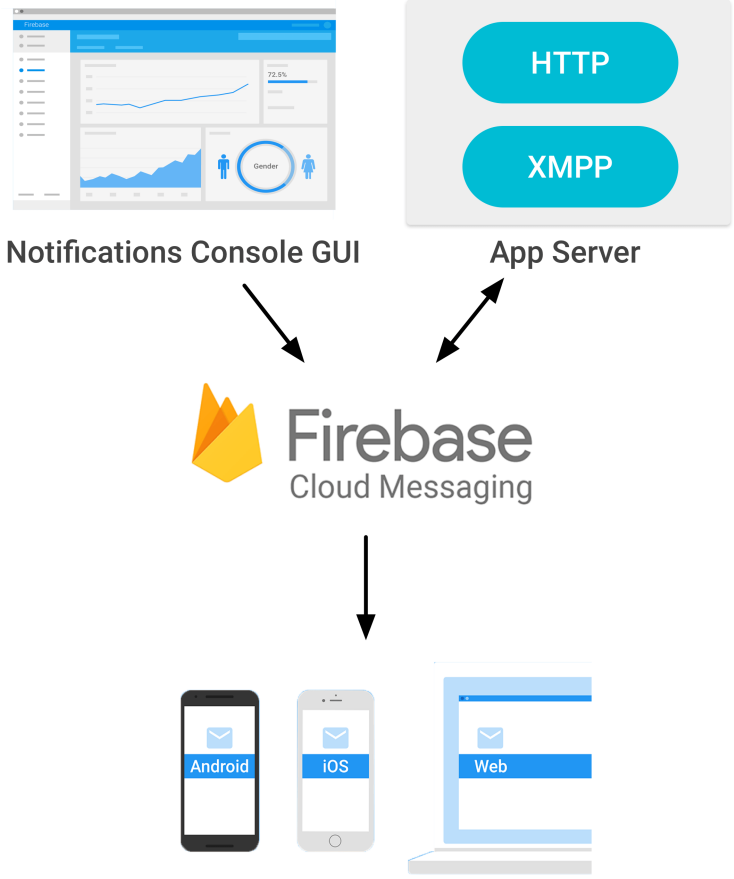
FIREBASE CLOUD MESSAGING

- Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost.
- Using FCM, you can notify a client app that new email or other data is available to sync.
- An FCM implementation includes an app server that interacts with FCM via HTTP or XMPP protocol, and a client app. You can compose and send messages using the app server or the Notifications console.
- Firebase Notifications is built on Firebase Cloud Messaging and shares the same FCM SDK for client development. For testing or for sending marketing or engagement messages with powerful built-in targeting and analytics, you can use Notifications. For deployments with more complex messaging requirements, FCM is the right choice.
- Server key in ‘Firebase Console > Settings > Cloud Messaging’

FIREBASE CLOUD MESSAGING

- ✧ Implementation path:

1. Set up the FCM SDK. Set up Firebase and FCM on your app according the setup instructions for your platform.
2. Develop your client app. Add message handling, topic subscription logic, or other optional features to your client app. During the development, you can easily send test messages from the Notifications console.
3. Develop your app server. Decide which server protocol(s) you want to use to interact with FCM, and add logic to authenticate, build send requests, handle response, and so on. Note that if you want to use upstream messaging from your client applications, you must use XMPP.



SET UP ANDROID CLIENT

- Add to your app's manifest a service that extends FirebaseInstanceIdService to handle the creation, rotation, and updating of registration tokens. This is required for sending to specific devices or for creating device groups.

```
<service android:name=".FirebaseIDService">
    <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>
```

- On initial startup of your app, the FCM SDK generates a registration token for the client app instance
- The registration token may change when:
  - The app deletes Instance ID
  - The app is restored on a new device
  - The user uninstalls/reinstall the app
  - The user clears app data.

ACCESS THE DEVICE TOKEN

- When you need to retrieve the current token, call FirebaseInstanceID.getToken(). This method returns null if the token has not yet been generated.
- The onTokenRefreshcallback fires whenever a new token is generated, so calling getToken in its context ensures that you are accessing a current, available registration token.

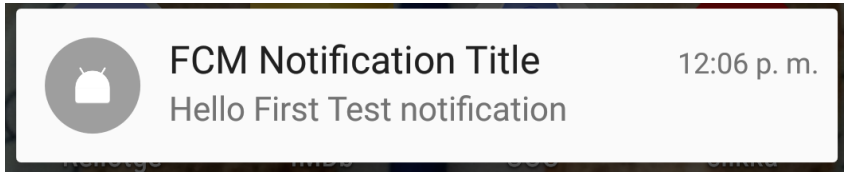
```
@Override
public void onTokenRefresh() {
    // Get updated InstanceID token.
    String refreshedToken = FirebaseInstanceId.getInstance().getToken();
    Log.d(TAG, "Refreshed token: " + refreshedToken);

    // If you want to send messages to this application instance or
    // manage this apps subscriptions on the server side, send the
    // Instance ID token to your app server.
    sendRegistrationToServer(refreshedToken);
}
```

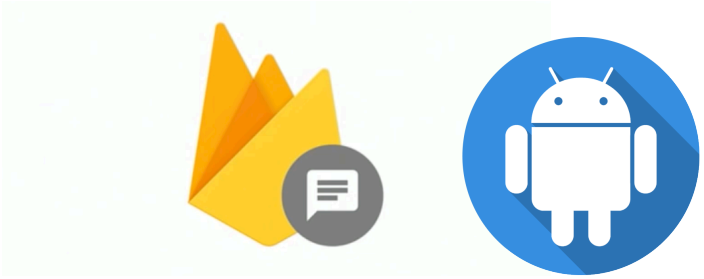
APP SERVER EXAMPLE

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;

import org.json.JSONObject;

public class FCMNotification {


       // Method to send Notifications from server to client end.
       public final static String AUTH_KEY_FCM = "API_KEY_HERE";
       public final static String API_URL_FCM = "https://fcm.googleapis.com/fcm/send";

       public final static String USER_DEVICE_TOKEN = "USER_DEVICE_TOKEN_HERE";
       public static void pushFCMNotification(String DeviceIdKey) throws Exception {

           String authKey = AUTH_KEY_FCM; // You FCM AUTH key
           String FMCurl = API_URL_FCM;

           URL url = new URL(FMCurl);
           HttpURLConnection conn = (HttpURLConnection) url.openConnection();

           conn.setUseCaches(false);
           conn.setDoInput(true);
           conn.setDoOutput(true);

           conn.setRequestMethod("POST");
           conn.setRequestProperty("Authorization", "key=" + authKey);
           conn.setRequestProperty("Content-Type", "application/json");
```

APP SERVER EXAMPLE

```
        JSONObject data = new JSONObject();
        data.put("to", DeviceIdKey.trim());

        JSONObject info = new JSONObject();
        info.put("title", "FCM Notification Title"); // Notification title
        info.put("body", "Hello First Test notification"); // Notification body
        data.put("notification", info);

        OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
        wr.write(data.toString());
        wr.flush();
        wr.close();

        int responseCode = conn.getResponseCode();
        System.out.println("Response Code : " + responseCode);

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();

        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();
    }

    @SuppressWarnings("static-access")
    public static void main(String[] args) throws Exception {
        FCMNotification obj = new FCMNotification();
        obj.pushFCMNotification(
                USER_DEVICE_TOKEN);
    }
}
```



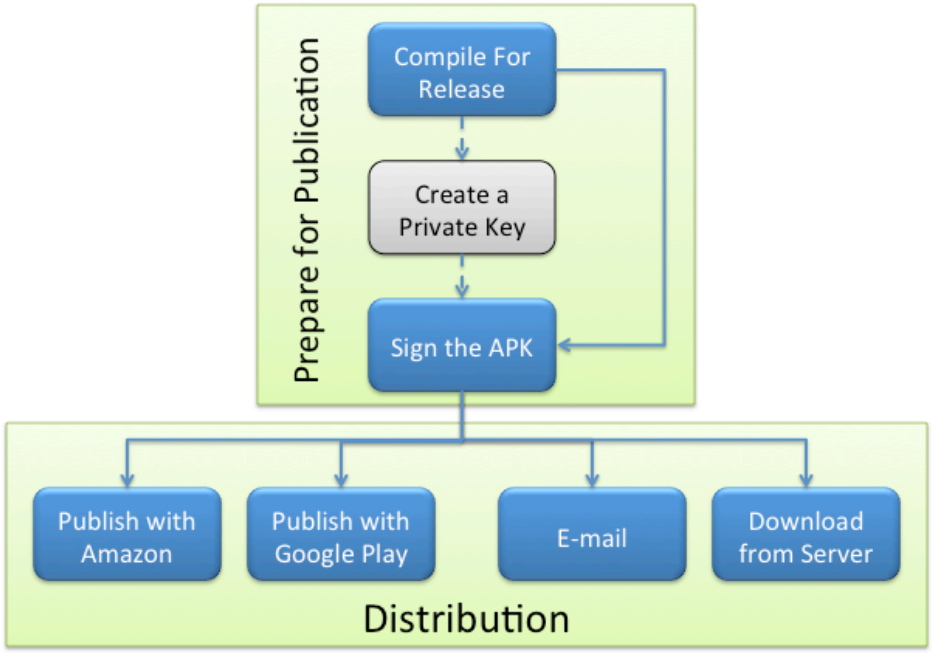


- Firebase notifications:
  - https://firebase.google.com/docs/notifications/
- Firebase console:
  - https://console.firebase.google.com
- Firebase Cloud Messaging Quickstart:
  - https://github.com/firebase/quickstart-android/tree/master/messaging
- Firebase Cloud Messaging:
  - https://firebase.google.com/docs/cloud-messaging/
- App server example:
  - http://stackoverflow.com/questions/38089148/send-push-notification-from-server-to-android-device-in-java

PUBLICATION

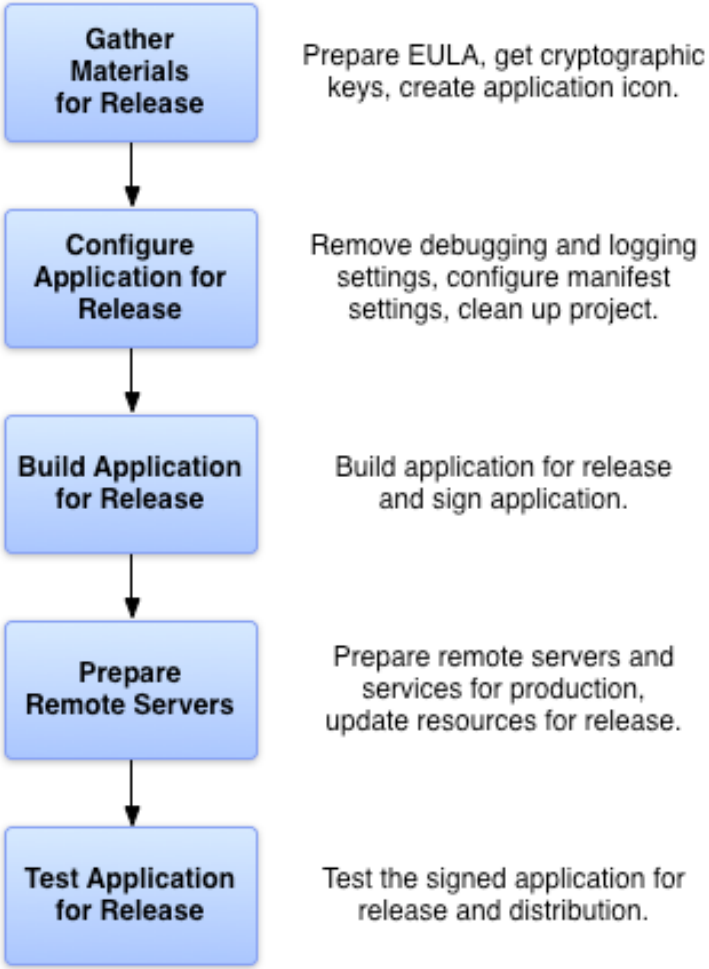
PUBLISHING APPS

- The final step in the development of an Android application is to publish the application.
- Publishing is the process of compiling the application so that it is ready for users to install on their devices, and it involves two essential tasks:
  - **Preparing for Publication** – A release version of the application is created that can be deployed to Android-powered devices.
  - **Distribution** – The release version of an application is made available through one or more of the various distribution channels.



PUBLICATION

PREPARE FOR RELEASE



PUBLICATION

GATHERING MATERIALS AND RESOURCES

- **Cryptographic keys**
  - The Android system requires that each installed application be digitally signed with a certificate that is owned by the application's developer (that is, a certificate for which the developer holds the private key).
  - The Android system uses the certificate as a means of identifying the author of an application and establishing trust relationships between applications.
- **Application Icon**
  - Be sure you have an application icon and that it meets the recommended icon guidelines. Your application's icon helps users identify your application on a device's Home screen and in the Launcher window.
  - If you are releasing your application on Google Play, you need to create a high resolution version of your icon
- **End-user License Agreement**
  - Consider preparing an End User License Agreement (EULA) for your application.
- **Miscellaneous Materials**
  - You might also have to prepare promotional and marketing materials to publicize your application.

CONFIGURING FOR RELEASE (I)

- **Choose a good package name**
  - Make sure you choose a package name that is suitable over the life of your application. You cannot change the package name after you distribute your application to users.
  - The package name can be set in application's manifest file.

- **Turn off logging and debugging**
  - Make sure you deactivate logging and disable the debugging option before you build your application for release.
    - You can deactivate logging by removing calls to Log methods in your source files.
    - You can disable debugging by removing the **android:debuggable** attribute from the <application> tag in your manifest file, or by setting the **android:debuggable** attribute to false in your manifest file.
  - Remove any log files or static test files that were created in your project.
  - Also, you should remove all Debug tracing calls that you added to your code, such as *startMethodTracing()* and *stopMethodTracing()* method calls.

CONFIGURING FOR RELEASE (II)

- **Clean up your project directories**
  - Clean up your project and make sure it conforms to the directory structure described in Android Projects.
  - Remove files that the application does not use (private or propietary data files, test libraries in /lib, and assets/ and res/raw files).

- **Review and update your manifest and Gradle build settings**
  - Verify that the following manifest and build files items are set correctly:
    - <uses-permission> element: You should specify only those permissions that are relevant and required for your application.
    - android:icon and android:label attributes: You must specify values for these attributes, which are located in the <application> element.
    - android:versionCode and android:versionName attributes: You must specify values for these attributes, which are located in the <manifest> element.
  - There are several additional manifest or build file elements that you can set if you are releasing your application on Google Play. For example, the android:minSdkVersion and android:targetSdkVersion attributes, which are located in the <uses-sdk> element.

PUBLICATION

PREPARE FOR RELEASE

- **Address compatibility issues**
  - Android provides several tools and techniques to make your application compatible with a wide range of devices:
    - **Add support for multiple screen configurations:** Make sure you meet the best practices for supporting multiple screens. Supporting Multiple Screens.
    - **Optimize your application for Android tablet devices**: Supporting Tablets and Handsets.
    - **Consider using the Support Library:** If your application is designed for devices running Android 3.x, make your application compatible with older versions of Android by adding the Support Library to your application project.

- **Update URLs for servers and services**
  - If your application accesses remote servers or services, make sure you are using the production URL or path for the server or service and not a test URL or path.

PUBLICATION

PREPARING FOR RELEASE

- After you finish configuring your application you can build it into a release-ready .apk file that is signed and optimized.
  - The JDK includes the tools for signing the .apk file (Keytool and Jarsigner):
  - The Android SDK includes the tools for compiling and optimizing the .apk file.
- If your application relies on a remote server, make sure the server is secure and that it is configured for production use.
  - This is particularly important if you are implementing in-app billing in your application and you are performing the signature verification step on a remote server.
- If your application fetches content from a remote server or a real-time service (such as a content feed), be sure the content you are providing is up to date and production-ready.

TESTING FOR RELEASE

- Testing the release version of your application helps ensure that your application runs properly under realistic device and network conditions.
  - Ideally, you should test your application on at least one handset-sized device and one tablet-sized device to verify that your user interface elements are sized correctly and that your app's performance and battery efficiency are acceptable.

- Common Android-related situations that you should consider when you test:
  - **Change in orientation:** Is the screen re-drawn correctly? Any custom UI code you have should handle changes in the orientation. Does the application maintain its state?
  - **Change in configuration:** You should test that the application updates itself to respond correctly to the new configuration.
  - **Battery life:** You need to write your app to minimize battery usage, test its battery performance, and test the methods that manage battery usage (Coding for Life -- Battery Life).
  - **Dependence on external resources:** If your application depends on network access, SMS, Bluetooth, or GPS, then you should test what happens when the resource or resources are not available.

PUBLICATION

SIGNING

- Android requires that all apps be digitally signed with a certificate before they can be installed.
  - Android uses this certificate to identify the author of an app, and the certificate does not need to be signed by a certificate authority.
  - Android apps often use self-signed certificates.
  - The app developer holds the certificate's private key.
- You can sign an app in debug or release mode.
  - You sign your app in **debug mode** during development and in release mode when you are ready to distribute your app. The Android SDK generates a certificate to sign apps in debug mode.
  - To sign apps in **release mode**, you need to generate your own certificate.

SIGNING

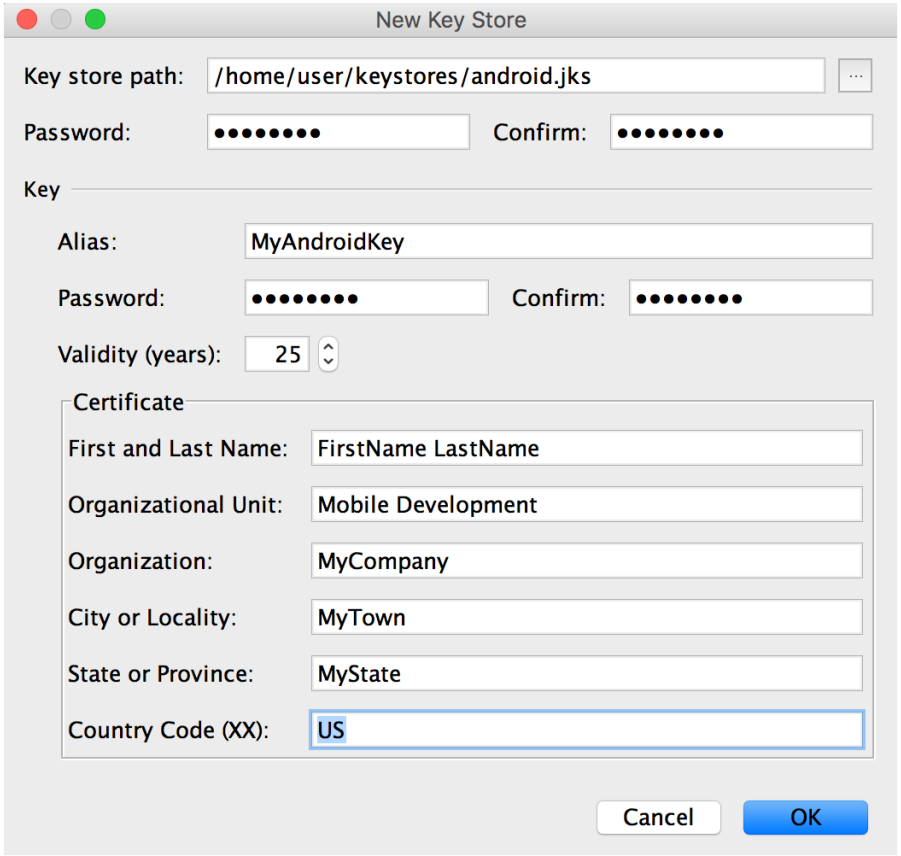
- In debug mode, you sign your app with a debug certificate generated by the Android SDK tools.
  - This certificate has a private key with a known password, so you can run and debug your app without typing the password every time you make a change to your project.
- Android Studio signs your app in debug mode automatically when you run or debug your project from the IDE.
  - By default, it uses a debug keystore, with a known password and a default key with a known password, located in $HOME/.android/debug.keystore.
- You can run and debug an app signed in debug mode on the emulator and on devices connected to your development machine through USB, but **you cannot distribute an app signed in debug mode**.

PUBLICATION

SIGNING FOR RELEASE

- You can use Android Studio to manually generate signed APKs, either one at a time, or for multiple build variants at once. Instead of manually signing APKs, you can also configure your Gradle build settings to handle signing automatically during the build process.

1. Invoke the “Build→Generated Signed apk” menu option.
2. Select the module.
3. Create a key store if you have not one.

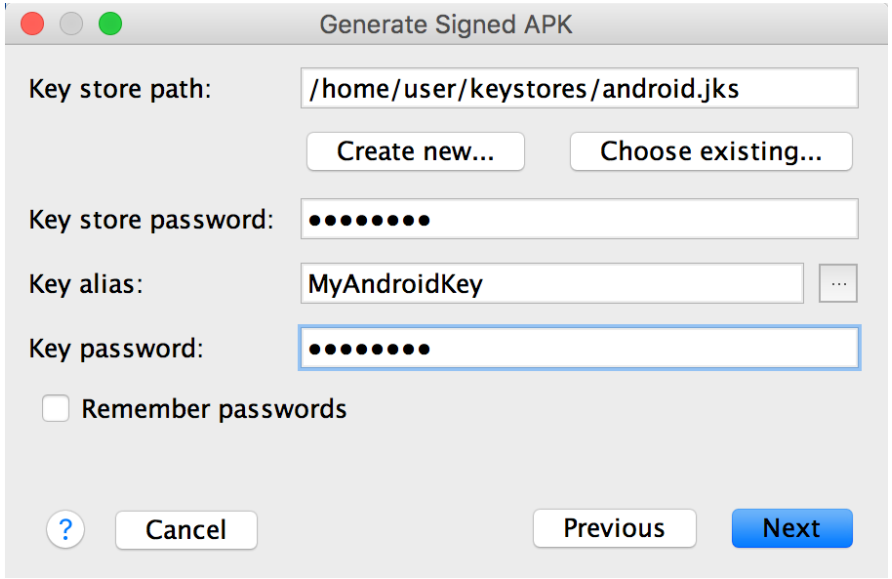


PUBLICATION

CREATING THE APK FILE

4. Select the keystore, a private key, and enter the passwords for both.
5. On the next window, select a destination for the signed APK(s).

✧ When the process completes, you will find your signed APK in the destination folder you selected above. You may now distribute your signed APK through an app marketplace like the Google Play Store, or using the mechanism of your choice.



✧ **Configure the build process to automatically sign your APK**. In Android Studio, you can configure your project to sign your release APK automatically during the build process by creating a signing configuration and assigning it to your release build type.

- ✧ Sign Your App: https://developer.android.com/studio/publish/app-signing.html

DISTRIBUTION

REGISTERING

- The first step in the application submission process is to create a Google Play Developer Console account. To do so, navigate to https://play.google.com/apps/publish/signup/ and follow the instructions to complete the registration process.
  - Note that there is a one-time $25 fee to register.
  - Once an application goes on sale, Google will keep 30% of all revenues associated with the application.

- Once the account has been created, the next step is to gather information about the application.

- Register for a Google Play Developer Console Account in order to bring your application to market. Some additional information will be required (Launch Checklist).

DISTRIBUTION

REQUIRED INFORMATION (I)

- **Title**: The title of the application.
- **Description**: Up to 4000 words describing the application.
- **Screenshots:** Up to 8 screenshots of your application running (a minimum of two is required). Google recommends submitted screenshots of the application running on a 7” or 10” tablet.
- **Language**: The language of the application (the default is US English).
- **Promotional Text:** The text that will be used when your application appears in special promotional features within the Google Play.
- **Application Type:** Whether your application is considered to be a game or an application.
- **Category:** The category that best describes your application (for example finance, health and fitness, education, sports etc.).

DISTRIBUTION

REQUIRED INFORMATION (II)

- **Locations:** The geographical locations into which you wish your application to be made available for purchase.
- **Contact Information:** Methods by which users may contact you for support relating to the application. Options include web, email and phone.
- **Pricing & Distribution**: Information about the price of the application and the geographical locations where it is to be marketed and sold.