Carma Firebase Integration Guide.

Rev pA3.

About this document

This document describes how to integrate Carma with Google's firebase push notifications for android. Firebase can also send iOS notifications but that is outside the scope of this document.

All instructions on building the android app assumes that you are using Android Studio 2.3+.

Change notes:

pA1	first version
pA2	minor review changes
pA3	added changes for opening app from notification
pA4	added apptype for firebase

Contents

How Firebase Cloud Messaging (FCM) Works

Firebase is backend platform for building Web, Android and IOS applications. It offers real time database, different APIs, multiple authentication types and hosting platform. This document covers sending push notifications to android mobile using Firebase.

Google's system considers three parties:

- The client application
- Google's FCM servers
- A 3rd party server, in this case Carma.

When a user installs an application, it must register itself to enable FCM. Once done, as soon as you decide to send your users a notification, a succession of actions are triggered:

- Carma sends the push notification to Google's FCM servers.
- These servers relay the message to all your registered mobile applications.
- Messages are queued and stored for devices that are offline.
- As soon as a device comes back online, FCM servers relay the queued message.
- The messages are received and presented according to the platform-specific implementation.

Integration

The steps for integration are as follows:

Before you begin coding

1. The app is registered with google and is given an appName
2. The app is registered in Carma using the appName and the Firebase Server Key.

The app code

1. The app requests a token from FCM
2. The App sends this token to Carma along with the appName.
3. The token is stored in Carma and associated with the app

Carma UI

1. A new Campaign is created as a push
2. The campaign is connected to the registered App.
3. The campaign is sent to google who will forward it to the registered devices. This process is not covered in this document. Please check http://expertise.carmamarketinghub.com/edit-content/edit-email-or-app-push-content/create-apppush/ for more information

Android App

Before you begin, you will need to have an android app that you want to send push notifications to.

This push notifications tutorial assumes that Android Studio is used as the IDE, with a target device running Android 4.0.4 or higher.

Firebase Cloud Messaging (FCM)

FCM is a service provided by Google that helps developers implement push notifications in their applications. By using FCM, developers are not required to implement their own method for sending data from their server to the client applications.

FCM Overview

Your App server -> FCM -> device running your client application

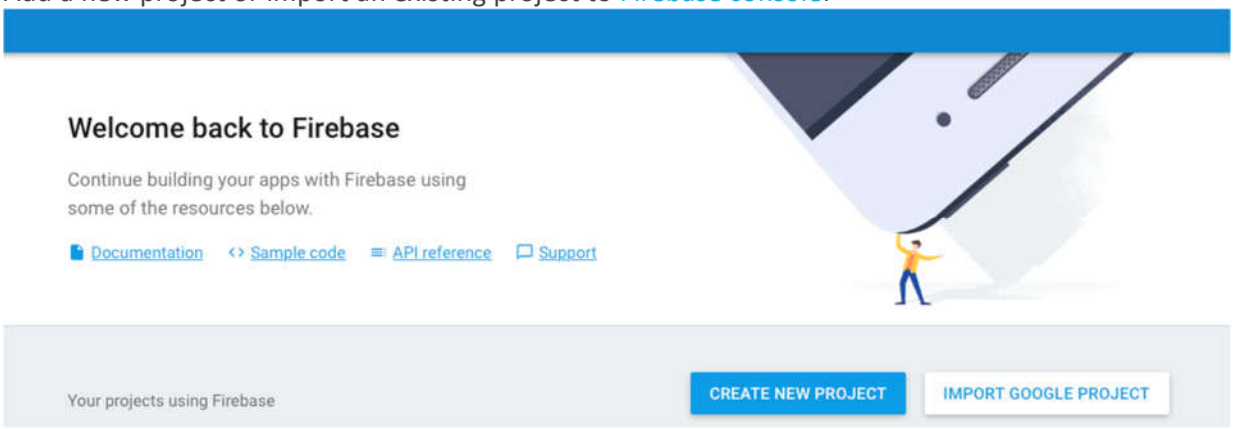
Both the app server and Android client need to register with FCM and provide information to uniquely identify and authorize them. Your Android app will need a Package name and your server will need a Web API key, both of which can be obtained using the Firebase Developers Console (https://console.firebase.google.com).

Configuring the Project in Fire Developers Console

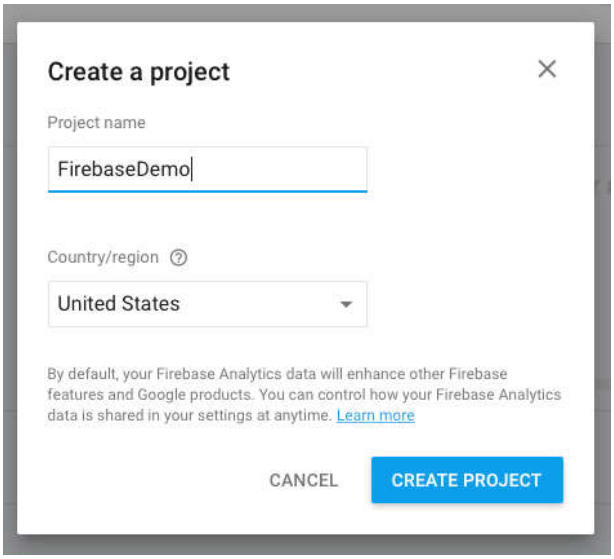
Below is a walkthrough of the steps you need to take to integrate firebase notifications with your app. There is also a good tutorial on Firebase you may want to read before starting: https://firebase.google.com/docs/android/setup

Get started

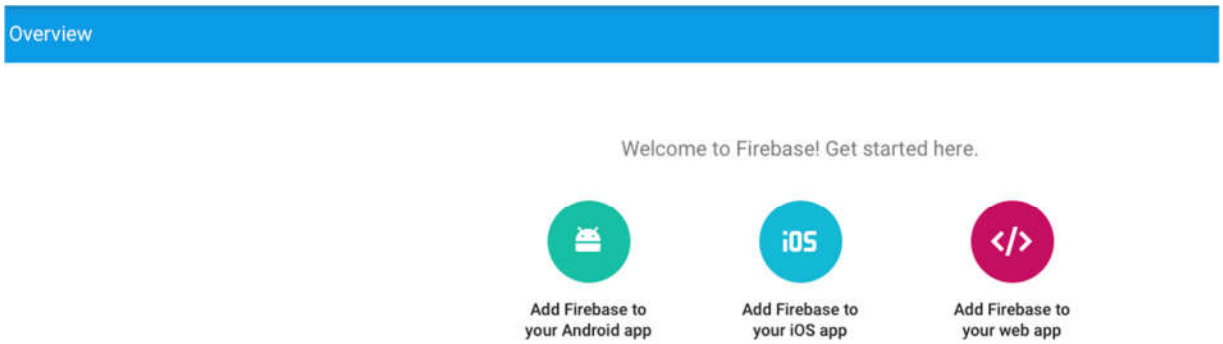
Add a new project or import an existing project to Firebase console.



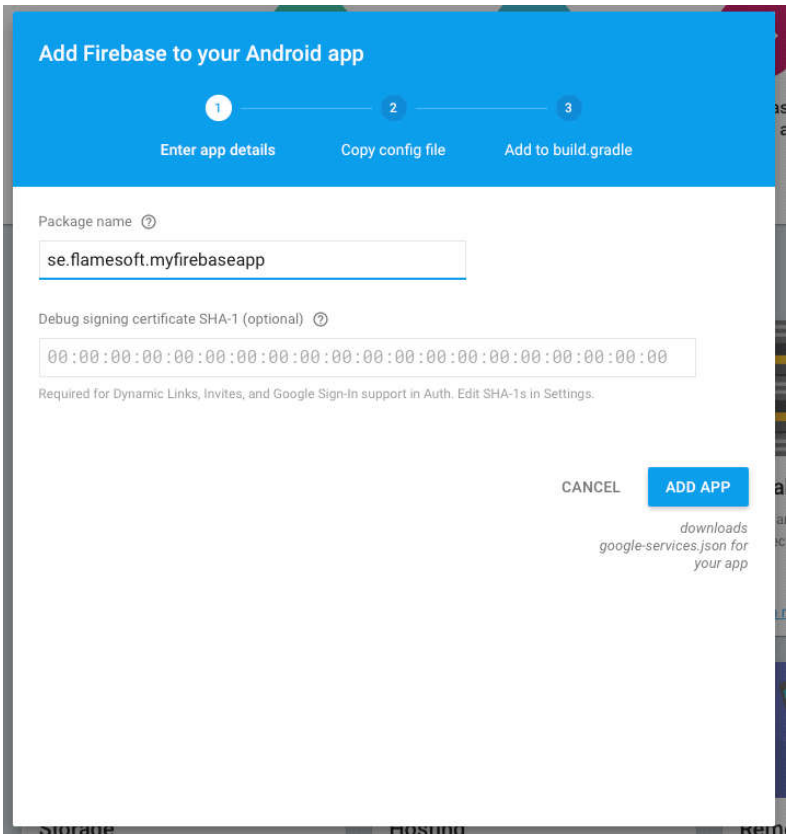
If you choose to create a new project, you need to set the project name and country. In this example, the project will be called *FirebaseDemo*.



Then select "Add Firebase to your Android app".



Set a package name for your app. SHA-1 is only used if you use firebase for authentication.



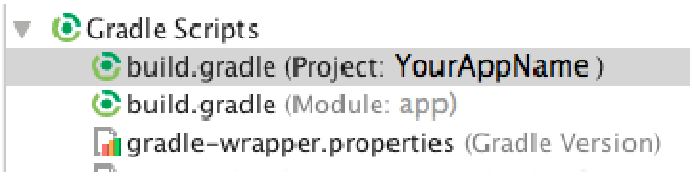
Click the **ADD APP** button here to download google-services.json. This is an important file and you will need to put it into your app.

Add google-services.json to your app folder

Replace the google-services.json in your app folder. The Google services plugin for Gradle will load the google-services.json file you just downloaded.

Configure gradle files

Open Android Studio and modify your build.gradle files to use the Google services plugin.



Update the project-level build.gradle (the one in your project folder)

Add the following line to the build.gradle file:

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

Update the app-level build.gradle (the one in your project/your app-module

Add this line to the bottom of the build.gradle file

```
Apply plugin: 'com.google.gms.google-services'
```

Add Firebase related dependencies

And Firebase related dependencies under dependencies in the same build.gradle file.

```
dependencies {
    compile 'com.google.firebase:firebase-core:11.4.2'
// this line must be included to integrate with Firebase
    compile 'com.google.firebase:firebase-messaging:11.4.2'
// this line must be included to use FCM
}
```

Update services using com.google.android.gms:play-services

If you add Firebase into an existing project which uses any function of gms:play-services, such as gps location, you have to update their versions, too. Upon writing this tutorial, 9.2.0 works well. If you get compilation problems, you need to check find out the correct version number.

```
  compile 'com.google.android.gms:play-services-location:11.4.2'
  compile 'com.google.android.gms:play-services-places: 11.4.2'
```

(d) Add the applicationId to the defaultConfig section

```
android {

   defaultConfig {
      applicationId "com.example.my.app"  // this is the id that your app has
   }
}
```

Add services to your app

Two services should be added to use Firebase Cloud Messaging service: a basic code for testing if push notification works, and other codes to handle receiving message when you app is in the foreground or sending message in your app according to your design. You must also implement code that will receive notifications when the app is in the background.

Add a service that extends FirebaseMessagingService

To be able to receive any notification in your app, you should add a service which extends

FirebaseMessagingService like this:

```
public class MyFirebaseMessagingService extends
FirebaseMessagingService {
 private static final String TAG = "FCM Service";
 @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        // Check if message contains a notification payload.
      if (remoteMessage.getNotification() != null) {

             // retreive rich content Url.
            String dataUrl = remoteMessage.getData().get("url");
            // Use pushUrl to mark the message as read if you do not use rich
            content.
            String pushReadUrl = remoteMessage.getData().get("pushreadurl");
            // Read custom data with a get on the applicable Key e.g.
            // String customData = remoteMessage.getData().get("someKey");

      }
.        Log.d(TAG, "From: " + remoteMessage.getFrom());
        Log.d(TAG, "Notification Message Body: " +
remoteMessage.getNotification().getBody());
    }
}
```

Then add it into the AndroidManifest.xml file.

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

Add a service that extends FirebaseInstanceIdService

```
public class FirebaseIDService extends FirebaseInstanceIdService {
    private static final String TAG = "FirebaseIDService";

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

        // TODO: Implement this method to send any registration to
your app's servers.
        sendRegistrationToServer(refreshedToken);
    }
/**
     * Persist token to third-party servers.
     *
     * Modify this method to associate the user's FCM InstanceID
token with any server-side account
     * maintained by your application.
     *
     * @param token The new token.
     */
    private void sendRegistrationToServer(String token) {

       // this is where the code for registering with Carma should go.
    }
}
```

Add it into the AndroidManifest.xml file, this makes sure that the service is loaded

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

Add code to receive notifications in the background.

If your app is in the background when a notification is received the app will start when you click on it.

This will start the apps MainActivity and the notification data will be available to the code.

In you main activity you must have code that read the notification data. The code below will retrieve that data.

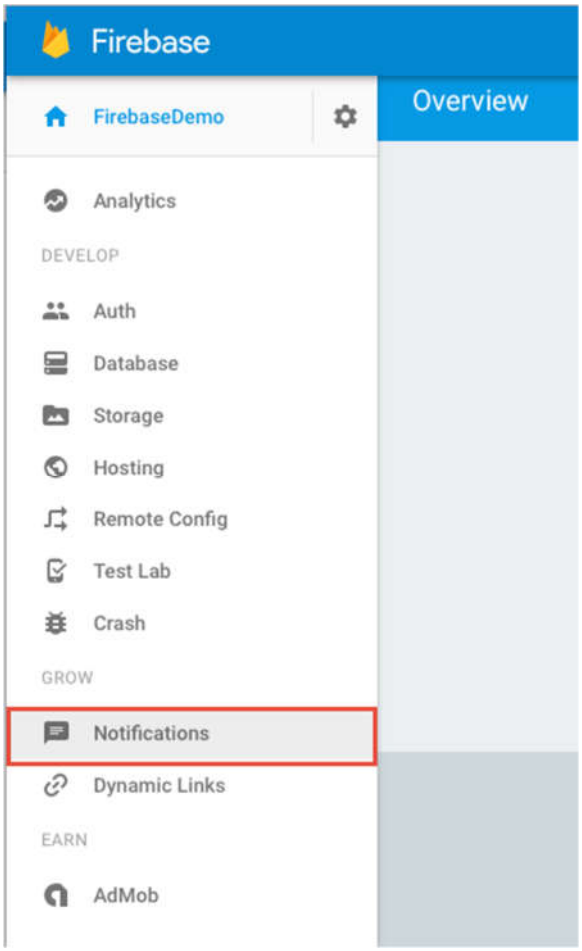
```
// handle push notification
String url = null;
String pushreadurl = null;
String collapse_key = null;
String somekey = null;
Bundle bundle = getIntent().getExtras();
if (bundle != null) {
    url = bundle.getString("url");
    pushreadurl = bundle.getString("pushreadurl");
    collapse_key = bundle.getString("collapse_key");
    somekey = bundle.getString("somekey");
    //bundle contain all info sent in "data" field of the notificationm i.e. all
custom and carma specific fields
}
```

You must also add an <intent-filter> to the <activity> of the main activity in the AmdroidManifest.xml file:

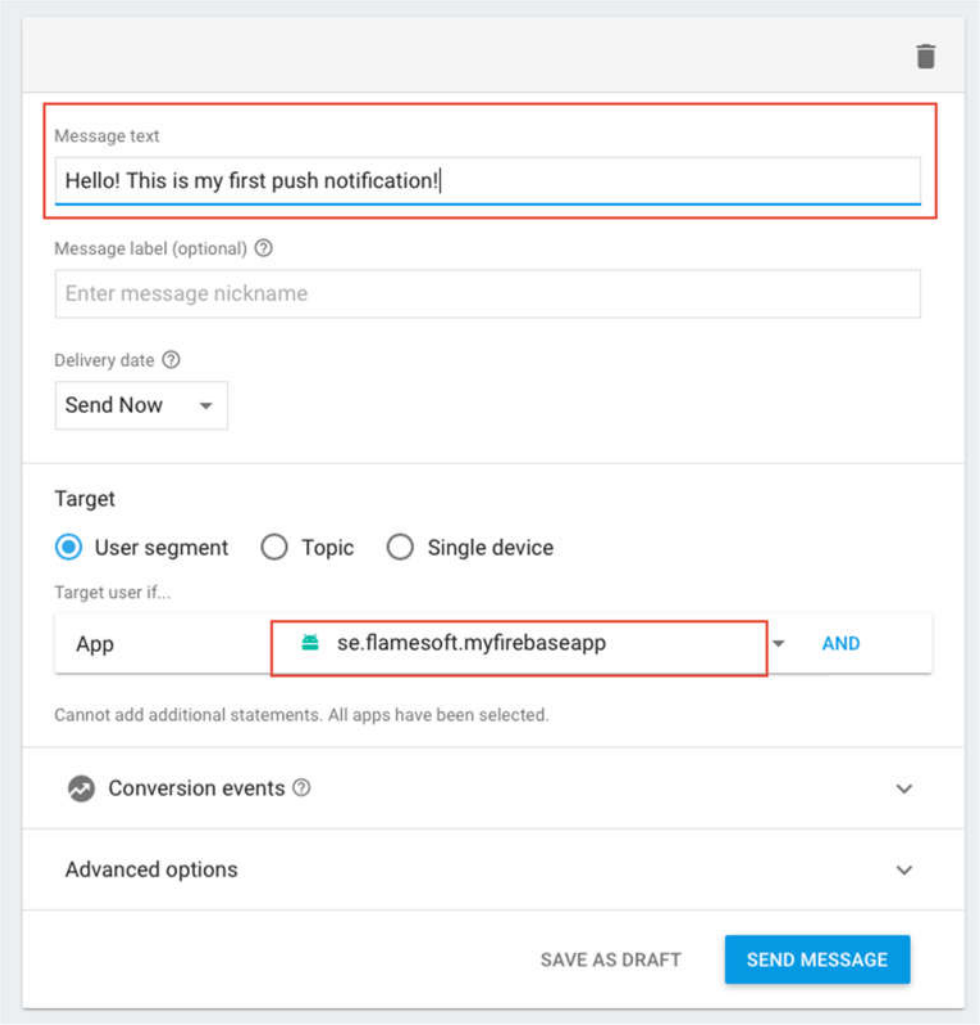
```
<intent-filter>
    <action android:name=".MainActivity" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Test and send your first push notification

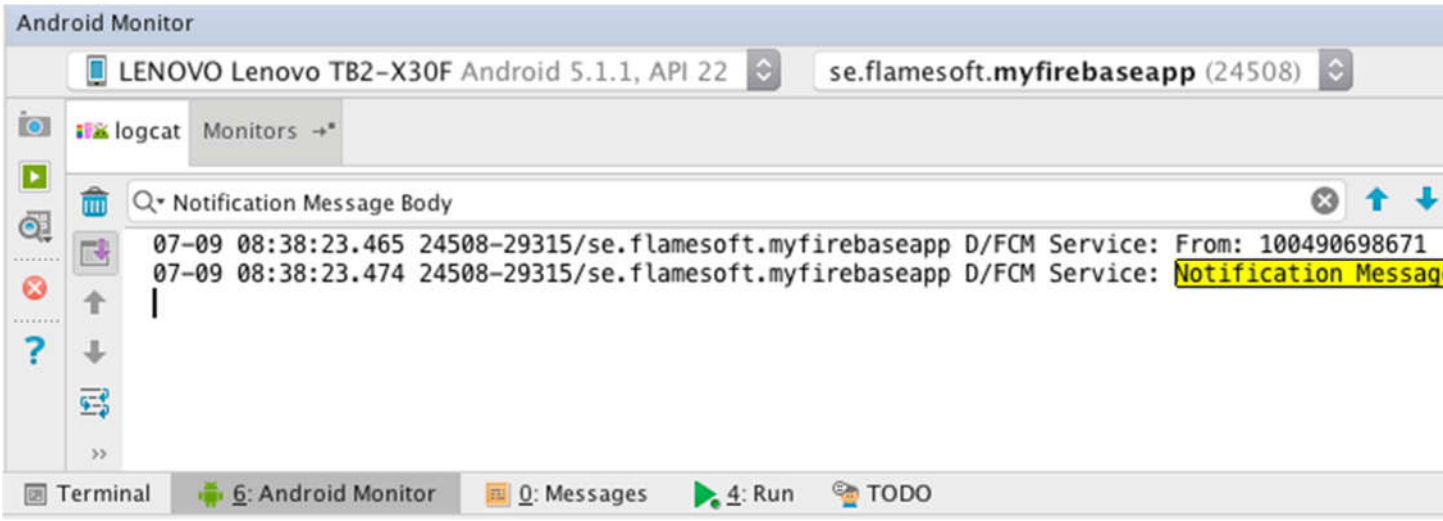
To see if the setup works, run a test by sending a test message to your own mobile. You can do this using the firebase console before you test to send a notification from Carma.


Firebase
FirebaseDemo
Overview
Analytics
DEVELOP
Auth
Database
Storage
Hosting
Remote Config
Test Lab
Crash
GROW
Notifications
Dynamic Links
EARN
AdMob

Write down your message and choose an app. Click "SEND MESSAGE".



Now you should get a push notification on your Android mobile. If your app is running on the background, you will get it on the mobile's notification center; otherwise you can see it in your Android Monitor log (we have to put a code to log incoming messages) like this.



If the setup is successful, you should get a notification on your mobile. Sometimes, it can take a couple of minutes for the message to send and arrive, so just be patient for a little while.

Write the Carma Specific Code

Carma credentials

The code above is generic for any push notification implementation. There are some additional tasks you need to perform in order to integrate you App with Carma.

Before you start coding you will need to find four pieces of data that are used in all interactions with Carma.

The first is the android PackageName i.e. `com.somecompany.app (appName)`

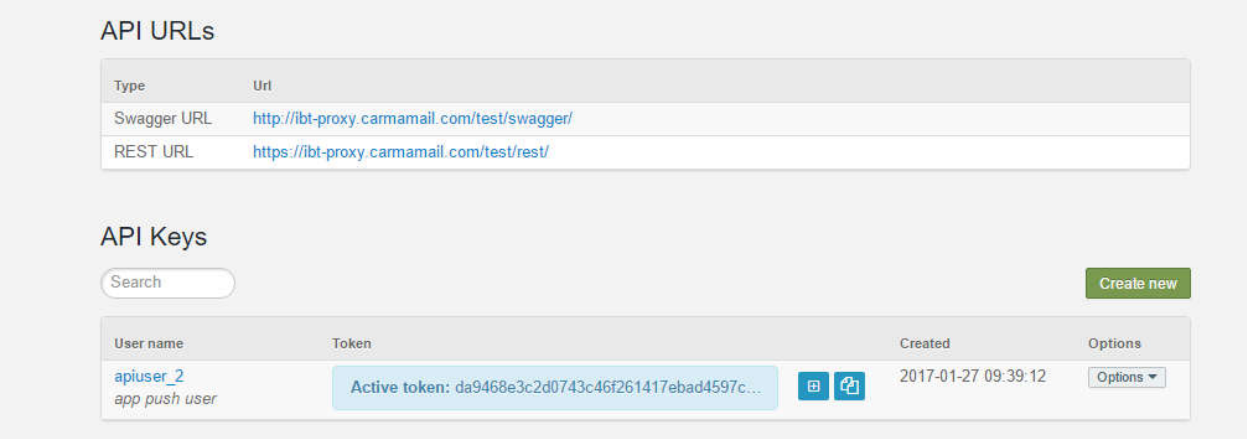
To find the other three you will need to logon to Carma. The data is:

- REST URL (`restUrl`)
- Carma API Key. (`carmaAccessToken`)
- Customer ID (`customerId`)

The first two can be found in Carma under Account Settings-> Carma API

(https://web-ibt-test.carmamail.com/carma/sv-se/carmaapi)

If you haven't created an API key before, click the *Create New* button to create one now.



The customer ID is found on the bottom of the main menu in Carma.

The usage of this information is described in a chapter below.

Request headers

All calls to carma need three request headers.

The resource uses a custom header for authentication and need you to set the *Content-Type* and *Accept* headers to "*application/json*"

```
    HttpPut post = new HttpPut(url);
    post.setEntity(se);
    post.setHeader("Accept", "application/json");
    post.setHeader("Content-Type", "application/json");
    post.setHeader("X-Carma-Authentication-Token", accessToken);
```

Registration

We can now begin coding. The first thing we need to do is to send the firebase token to Carma. The token is received in the `FirebaseIDService` that is shown further up in the document.

All call to Carma are REST calls. Java have a number of ways to implement REST, so you will need to choose one to your liking. The example below uses a bare metal method you to give some example on how this could be implemented.

This token needs to be registered in Carma along with the appId and an originalid of you choosing. The originalid is an identifier of the user in you system. It could be an email address or a primary key of the user in your database. If you do not have an originalid for the current user in your app, you may use the token as originalId. You register by issuing a PUT request to:

```
https://<serverUrl>/rest/<customerid>/apps/<appid>/pushdevices
```

with a json payload containing at the minimum:

- deviceToken
- originalId

NOTE! Even if they are the same both values must be provided.

A minimal version of the payload would look like this:

```
{
  "originalId": "abcabc123123",
  "deviceInfo": [{"devicetoken": "abc123abc123kmlkmlkml"}]
}
```

NOTE! If you use the same appId for apps of different platforms e.g. com.company.name for both iOS and Android, you must add the appInfoId to the token. The appInfoId can be found in carma under *Account settings-> Push Apps.*

Valid values for appType are:

- 1 : iOS
- 2: Android
- 3: Windows
- 6: Firebase

The payload will then be:

```
{
 "originalId": "abcabc123123",
 "deviceInfo": [{"devicetoken": "abc123abc123kmlkmlkml", "appType": "1",
"sandboxToken": "false"}]
}
```

The payload will then be:

```
{
 "originalId": "abcabc123123",
 "deviceInfo": [{"devicetoken": "abc123abc123kmlkmlkml", "appInfoId": "42"}]
}
```

In this payload you can also add other information about the user that should be transferred to Carma. Below is the full list of available properties.

```
{
  "listId": 1000000005,
  "country": null,
  "originalId": "abcabc123123",
  "firstName": "lars",
  "lastName": "hansson",
  "middleName": null,
  "emailAddress": "lars@compost.se",
  "title": null,
  "dateOfBirth": null,
  "city": null,
  "zipcode": null,
  "sex": null,
  "mobileNumber": null,
  "optOutDate": null,
  "dateOfInvalidation": null,
  "optOutMobileDate": null,
```

```
  "active": true,
  "properties": {
    "food": "meat",
    "drink": "tea"
  }
,
  "deviceInfo": [
    {
      "devicetoken": "abc123abc123kmlkmlkml",
      "manufacturer": "apple",
      "model": "iphone 6",
      "osVersion": "9.2",
      "country": "Sweden",
      "dateOfInvalidation": null,
       "appInfoId": 42,
      "invalidationType": 0
    }
  ]
}
```

The same struct is used to return info on the device, but it is then appended with all devices that is connected to the contact.

The following code can be used for the registration:

```
private void registerDeviceAndProfile(CarmaUserProfile profile) {

    HttpClient client = new DefaultHttpClient();
    JSONObject parent = new JSONObject();
    JSONArray deviceInfo = new JSONArray();
    JSONObject token = new JSONObject();
       String packageName ="com.your.packagename";
       String accessToken ="234kj5h345jh3kj532kjh5g35345";

    try {
        token.put("devicetoken", profile.getToken());
        deviceInfo.put(token);
        parent.put("deviceInfo",deviceInfo);
        parent.put("originalId", profile.getOriginalId());
        parent.put("emailAddress",profile.getEmail());
        parent.put("mobileNumber",profile.getMobile());
    } catch (JSONException e) {
        e.printStackTrace();
    }
    final String url = Constants.server + "/rest/" +new
Integer(Constants.customerId).toString() + "/apps/"+packageName+"/pushdevices";
    StringEntity se;
    try {
        se = new StringEntity( parent.toString(),"UTF8");
        se.setContentType(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
        se.setContentEncoding("UTF8");

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        return;
    }
    HttpPut post = new HttpPut(url);
```

```
    post.setEntity(se);
    post.setHeader("Accept", "application/json");
    post.setHeader("Content-Type", "application/json");
    post.setHeader("X-Carma-Authentication-Token", accessToken);
    try {
        final HttpResponse response = client.execute(post);
        StatusLine line = response.getStatusLine();
        final int code = line.getStatusCode();
        StringBuilder sb = new StringBuilder();
        try {
            BufferedReader reader =
                    new BufferedReader(new
InputStreamReader(response.getEntity().getContent()), 65728);
            String cline = null;
            while ((cline = reader.readLine()) != null) {
                sb.append(cline);
            }
        }
        catch (IOException e) { e.printStackTrace(); }
        catch (Exception e) { e.printStackTrace(); }
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (Exception e) {
    }
}
```

Invalidation of a devicetoken.

If you want to remove a registration of a devicetoken for an OriginalId you can issue a request to

```
https://<serverUrl>/rest/<customerid>/apps/<appid>/pushdevices/invalidation
```

with a json payload containing at the minimum:

- deviceToken
- originalId,

```
{
    "deviceInfo":[{"devicetoken": "abc123abc123kmlkmlkml_2"}],
    "originalId":"abcabc123123"
}
```

NOTE! Even if the originalId and devicetoken are the same, both values must be provided.

Retrieving information about a contact

If you want to retrieve the information connected to a contact you can issu a GET request to

```
https://<serverUrl>/rest/<customerid>/apps/<appid>/pushdevices/<originalId>
```

You will receive the same struct as when you register a device. It will contain a list of all devices registered on the originalId. If you have imported information by other means i.e, via scheduledimport on the list, this data will also retrieved.

Receive Push

A Carma push message contains the following fields:

- alert - intended to be the line shown next to the icon in the notification
- body - intended to be the line shown under the first in the notification
- url - the url pointing to the actual rich content of the push
- pushreadurl - call this URL to mark the push as read in Carma.
- source - will always by “carma”. This should be used to distinguish between push messages that comes from other systems than carma. A game for example might have high-score system in place that will notify a player
- any other custom name/value pair that you have supplied in the carma UI för the push.

It’s important to remember that it’s up to the application developer to decide how these fields should be used.

Our Android client is now registered with FCM, and our server can begin sending messages to devices running our client using the provided API Server key and FCM registration token. Messages for our Android application are received by the CarmaFMSListenerService

that we declared in our Manifest. When a message is received from our app server, the FcmReceiver will start our CarmaFMSListenerService. This is where we write code to process the messages.

Load rich content

By looking for the extra parameter url when starting the main activity you can retrieve the link to rich content. The code below starts an activity AddWebView with the url as a parameter when a push is received.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Bundle extras = getIntent().getExtras();
    if ( extras != null) {
        String url = extras.getString("url");
        if (url != null) {
            Intent i = new Intent(getBaseContext(), AddWebViewActivity.class);
            i.putExtra("url", url);
            startActivity(i);
        }
    }
```

Below is the code for an AddWebView activity that shows the rich content in a new WebView

```
public class AddWebviewActivity extends AppCompatActivity  {

    private static final String TAG = ContentActivity.class.getSimpleName();
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_content);
        Bundle extras = getIntent().getExtras();
        String url = extras.getString("url");

        if (!TextUtils.isEmpty(url)) {
            WebView view = (WebView) findViewById(R.id.rich_content_webview);
```

```
            view.invokeZoomPicker();
            WebSettings settings = view.getSettings();
            settings.setJavaScriptEnabled(true);
            view.setScrollBarStyle(WebView.SCROLLBARS_OUTSIDE_OVERLAY);
            view.getSettings().setLoadWithOverviewMode(false);
            view.getSettings().setUseWideViewPort(false);
            view.setWebContentsDebuggingEnabled(false);
            try {
                view.loadUrl(url);
            } catch(Exception ex) {
                //Log.e(TAG, ex.getMessage());
            }
        }
    }
```

Carma Account

Setup App in Carma:

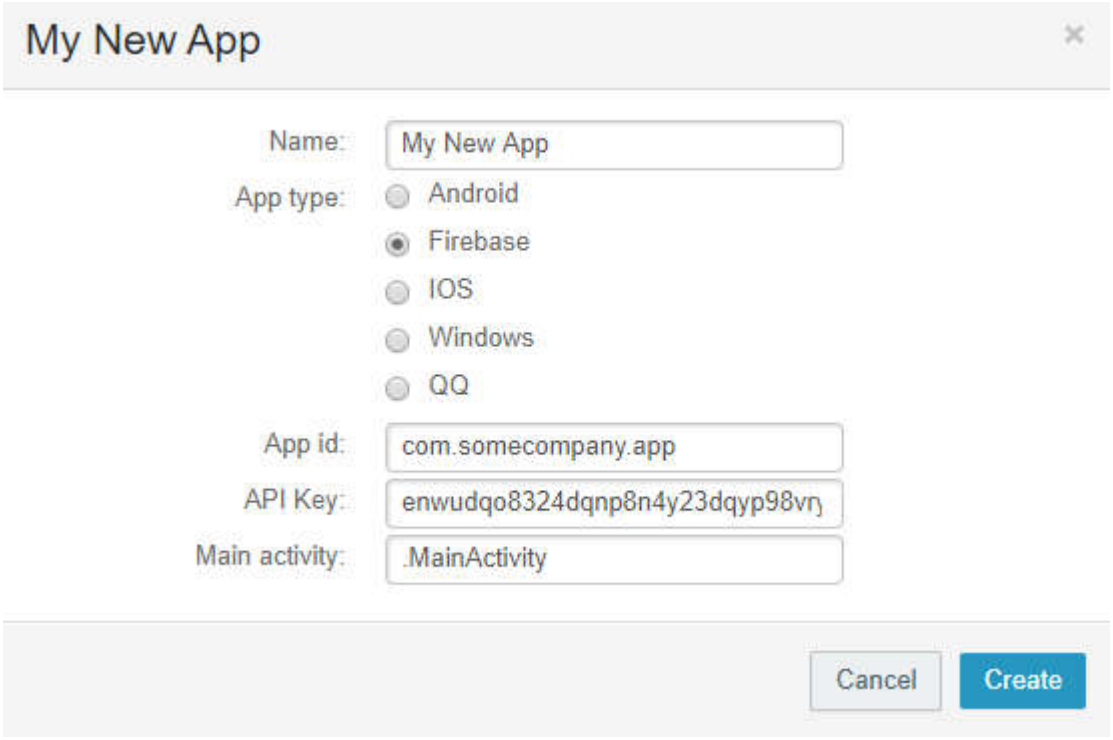
Click on A*ccount Setting* in the left hand menu and then choose *Push Apps* under *Manage*

Click **Create new** and you will see the following dialog where you enter the information

To register your app in carma you will need the name of you app, following the code above that would be

```
com.somecompany.app
```

the Web API Server key you retrieve from the Firebase developer console



Click *Create* and you are ready to start sending Push requests to you users.

The Rich content of the Campaign will be available to the push by calling the URL that is received in the url field in the notification.

Carma Content

The content of a push message is not sent in the push itself, but is located at server side where the url in message is pointing. Content is usually a page of html that is generated for the particular device, user by

a template of the server side. It could however be XML or other text-based formats as defined by template.

The most common way to handle content is to simply open an webview and pointing it to the url, making it display the page. The developer could however choose to handle it in different ways. Examples would be opening the device browser instead by an intent, downloading the content and parsing it to extract information that is used in the application somehow or passing it to another application.

NOTE! If you don't want to fetch the rich content you can mark the push as opened in carma by issuing a GET request to the *pushreadurl* that is a part of the notification payload.

Testing and troubleshooting

Once you have done integration, perform the following tests.

- Install the application on a new device
- Notice that the number of recipients increases for each time you do this. If not, check that the id's you supply are correct. Check also that the appid you gave when registering the application is the same as the package given in the top of your manifest.
- If the list does grow, test a simple push while the application is running. If a notification does not show up after a few minutes, make sure that you have given the correct api-key when registering the app in Carma.
- Once the above works correct, verify that the notifications show up when sending a push even though the application is not running.