HALO Server Integrations

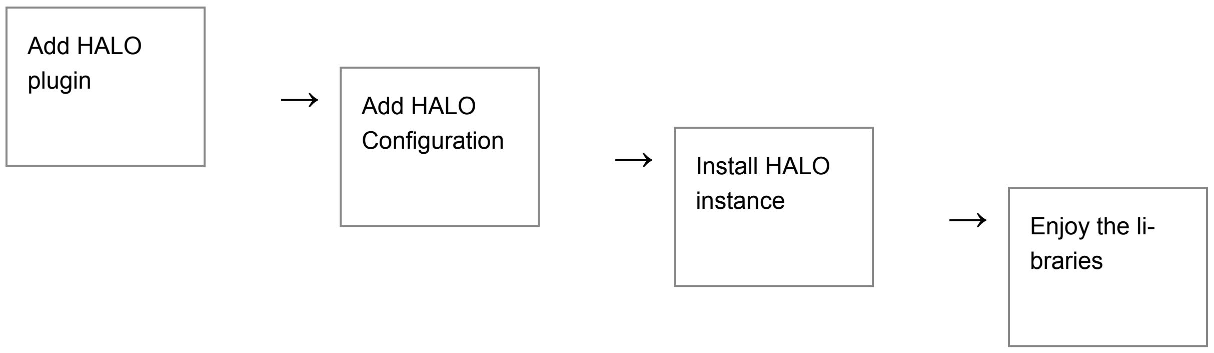
Version 2.5

Last generated: March 28, 2018

Table of Contents

SDK samples

Android SDK - Getting Started



This getting started guide will guide you on setting up HALO SDK for Android in a few minutes. We will provide a step by step guide to get everything working with the most basic setup, for more detailed information about specific calls or how a module works check the sidebar.

Warning: HALO SDK is now compatible with Android Plugin for Gradle 3.0.1+

Step 1: Add the HALO plugin

Open the build.gradle of the project root and add the plugin to the classpath:

```
buildscript {
    dependencies {
          classpath 'com.mobgen.halo.android:halo-plugin:{version}'
    }
}
```

Now apply the halo plugin to your HALO application after the android one:

```
apply plugin: 'halo'
```

Step 2: Add HALO configuration

Open the build.gradle of your app and apply the basic configuration based on your HALO project. Here you have the minimal configuration you will need. Put it after the Android node.

```
halo {
    clientId "YOUR_HALO_KEY"
    clientSecret "YOUR_HALO_SECRET"
    services {
        //you have to add this closure empty if you dont want any se
rvice
    }
}
```

Warning: If you want go in deep with all the options with the gradle configuration or variants configuration, please refer to the detailed documentation (page 9)

Step 3: Init the Halo instance inside your app

Option 1: I don't have a custom Application class

If you **don't** have a custom application class, follow this instructions. Open your AndroidManifest.xml and apply the following configuration to your Application node:

```
 <application
        ...
        android:name="com.mobgen.halo.android.sdk.api.HaloApplicatio
n">
        ...
</application>
```

Note: This will create the `Halo` instance for you. You can always access this instance by calling `HaloApplication.halo()`

Option 2: I do have a custom application class

In this case you can either install HALO by yourself or extend the `HaloApplication` class provided and override some of the methods that come with it.

Adding to your custom application class:

```
public class MyCustomApplication extends AnotherApplicationClass {
    @Override
    public void onCreate(){
        super.onCreate();
        Halo.installer(this).install();
    }
}
```

Extending the `HaloApplication` class:

```
public class MyCustomApplication extends HaloApplication {
    public Halo.Installer onCreateInstaller(){}
    public Halo.Installer beforeInstallHalo(@NonNull Halo.Installer
installer){}
    public Halo onHaloCreated(@NonNull Halo halo){}
}
```

Step 4: Start using HALO in your app

Use any of the HALO APIs or plugins without worrying when is it ready, since this is managed internally by the different libraries.

Android SDK - Installer Options

In the install step of HALO there are many parameters that can be configured. Here is the list of items and what are their purposes:

- **config**: provides the configuration for the HALO framework so you can add some aditional behaviour or configuration on it.
- **credentials**: sets the credentials client id and secret. They are added with gradle by default.
- **debug**: sets the debug flag to see what is going behind the scenes in HALO. See also the debug recipe (page 28).
- **printLogToFile**: if debug mode is enabled then you can print the log to file. See also the print log policies (page 28).
- **endProcess**: adds a startup process that will be run during the installation process in HALO.
- **addTagCollector**: adds a tag collector that allows you to add some tags in the moment the app is initialized.
- **enableDefaultTags**: adds the halo default tags into the tag collectors. This was enabled by default some time ago but now it is optional.
- **environment**: sets the environment that will be used for the calls of halo. See also server enviroment (page 26).
- **disablePinning**: disables the SSL pinning in the HALO SDK. See also SSL pinning (page 26).
- **enableServiceOnBoot**: enable the startup receiver to launch the service on foreground after a device reboot (only for android api 26+)
- **channelServiceNotification**: set the channel name and the icon for the notification if the service is running on foreground. If you don't set any channel nor icon you will get a Halo one (only for android api 26+)

Android SDK - Getting Started With Gradle Plugin

Here we provide the step by step guide to add the plugin to your project and start using it with the minimal configuration.

Warning: HALO SDK is now compatible with Android Plugin for Gradle 3.0.1+

1. Apply the classpath to the buildscript

Open the build.gradle of the project root and add the plugin to the classpath:

```
buildscript {
    dependencies {
          classpath 'com.mobgen.halo.android:halo-plugin:{pluginVers
ion}'
    }
}
```

2. Apply the plugin

Now apply the configuration plugin to your HALO managed application project after applying the android application plugin.

```
apply plugin: 'halo'
```

Note: You can also use the name ‘com.mobgen.halo.android’ for the plugin if you want to.

Tip: The plugin can only be applied to android application projects, don’t apply it to any android library project.

3. Add the HALO configuration

Open the build.gradle of your app and apply the basic configuration based on your HALO project. This is the minimal configuration you have to add to the plugin. If you add less information than that the plugin will not allow to compile.

```
halo {
    clientId 'YOUR_HALO_KEY'
    clientSecret 'YOUR_HALO_SECRET'
    services {
        //you have to add this closure empty if you dont want any se
rvice
    }
}
```

Warning: If you want go in deep with all the options with the gradle configuration or variants configuration, please refer to the detailed documentation (page 9)

Android SDK - Gradle Plugin Options

Gradle plugin options

What does the plugin do?

To avoid lines and lines of configuration we decided to create a gradle plugin to make your life as a developer easier. It handles the dependencies and the magic numbers you need to access HALO.

In your project you need to use Android Studio, or at least, a gradle build system. With that in mind we will describe how to put this plugin in your application project and the possible options to customize the SDK we are offering.

Options

In the HALO configuration inside the build.gradle there are many options you can configure to keep the SDK configured as you need. Here you have the reference for each property:

clientId

Client id of the application created in HALO. Its a required string you can find it in the CMS under apps section.

clientSecret

Client secret of the application created in HALO. Its a required string you can find it in the CMS under apps section.

clientIdDebug

Client id of the application created in HALO for testing purposes, this will be used when the debug flag is set in the installer. Its a optional string you can find the string in the CMS under apps section.

clientSecretDebug

Client secret of the application created in HALO for testing purposes, this will be used when the debug flag is set in the installer. Its a optional string you can find the string in the CMS under apps section.

services

Allows you to add more services from HALO. Only when you enable a service the dependencies will be automatically imported. This closure is mandatory also if you dont want to enable any service. In the following table we list all the service available:

Service	**Description**
analytics (Boolean:Optional)	Enables the analytics library for HALO SDK.
auth (Closure:Optional)	Enables the authentication library for the HALO SDK. This closure should contain **google** or **facebook** optional strings with each Google and Facebook api credentials.
content (Boolean:Optional)	You will enable the content library. This library helps when retrieving content.
notifications (Boolean:Optional)	Enables Google FCM integration with HALO used to receive push notifications. Remember to add also the google-services.json to your project.
presenter (Boolean:Optional)	Enables the presenter UI library for HALO. It is a small helper to use the MVP pattern in the UI.
translations (Boolean:Optional)	Enables the translations library for HALO.
twofactorauth (Closure:Optional)	Enables the two factor authentication library for HALO. You will have to use **sms** or **push** boolean to enable one or both services under this closure.

If you want to have both credentials for debug and prod and the following services enabled (twofactor, notifications, translations, content and auth) you have to provide the following HALO closure:

```
halo {
    clientId "halotestappclient"
    clientSecret "halotestapppass"
    clientIdDebug "halotestappclientdebug"
    clientSecretDebug "halotestapppassdebug"
    services {
        twofactorauth {
            push true
            sms true
        }
        notifications true
        translations true
        content true
        analytics false
        auth {
            google "google_credential"
            facebook "facebook_credential"
        }
    }
}
```

In the case you want different configurations for different variants, you can enable it with this configuration. You have to put the same configuration with services and clientId/clientSecret as you do in the global config. Both kind of configurations cannot be mixed. In this example we have three differente flavours: dev, qa and prod:

```
halo {
    androidVariants {
        dev {
            clientId "YOUR_HALO_KEY"
            clientSecret "YOUR_HALO_SECRET"
            services {
                //you will need to apply here all modules you want t
o import
            }
        }
        qa {
            clientId "YOUR_HALO_KEY"
            clientSecret "YOUR_HALO_SECRET"
            services {
                //you will need to apply here all modules you want t
o import
            }
        }
        prod {
            clientId "YOUR_HALO_KEY"
            clientSecret "YOUR_HALO_SECRET"
            services {
                //you will need to apply here all modules you want t
o import
            }
        }
    }
}
```

Full working example copy/paste script

Here you can find a HALO plugin configuration you can put and fill in your project:

```
apply plugin: 'com.android.application'
apply plugin: 'halo'

android {
    ...
}

halo {
    clientId "halotestappclient"
    clientSecret "halotestapppass"
    clientIdDebug "halotestappclient"
    clientSecretDebug "halotestapppass"
    services {
        content true
        notifications true
    }
}
```

Here the same configuration, but with notifications enabled only for release builds.

```
apply plugin: 'com.android.application'
apply plugin: 'halo'

android {
    ...
}

halo {
    androidVariants {
        debug {
            clientId "TestDebug"
            clientSecret "TestDebug"
            services {
                content true
            }
        }

        release {
            clientId "TestRelease"
            clientSecret "TestRelease"
            services {
                content true
                notifications true
            }
        }
    }
}
```

Android SDK - HALO Core API

You will be able to access some internals of HALO by accessing to its `HaloCore` instance. This instance keeps the current device with its segmentation tags, the credentials, token and configuration provided in the plugin.

You can always access the core from the instance but be careful when modifying anything, since most of it is automatically assigned and you typically don't want to change the internals:

Method name	Functionality
`sessionManager()`	provides the session manager bucket where the HALO session is stored.
`debug()`	returns the current debug configuration.
`credentials()`	returns the current credentials stored in the core.
`credentials(credentials)`	sets the current credentials and reauthenticates the APIs.
`logout()`	if you are logged in with some credentials you will be logged out from those user credentials.
`version()`	provides the library version which is the version of all the items in the sdk.
`flushSession()`	removes the current session token, so another one will be requested when any request is done with Halo.
`framework()`	provides the instance of the HALO framework used internally to manage all the calls.
`manager()`	provides access to the `HaloManagerApi` which contains all the operations you can do to manage the internal state of the HALO instance.

Method name	Functionality
`device()`	provides the current stored device. If HALO is not fully installed the device will be an anonymous one not ready yet. This cannot be null.
`pushSenderId()`	provides the configured sender id for the push notifications. If it is null, the push notifications are not enabled.
`notificationsToken()`	provides the token from the Firebase cloud messaging service if the notifications library is enabled.
`notificationsToken(token)`	sets the FCM token. Just for internal use.
`segmentationTags()`	collects all the segmentation tags for the current device.
`serverVersionCheck()`	tells if the current sdk is outdated based on the server version. 1 means valid, 2 means outdated, 3 means not checked (ie, no internet).
`isVersionValid()`	checks the serverVersionCheck to ensure wether this version is valid or not. Valid means it has not been checked for some reason (typically network problems) or it is outdated.

Android SDK - HALO Manager API

API definition

The manager plugin is in charge to all the management actions that can be done in the core, such as changing the tags of the device or requesting the modules that belongs to the current application.

In this guide you will find an explanation of the api available methods. To create an instance for the manager API you should use:

```
Halo.core().manager();
```

Method name	**Functionality**
storage	Provides the current storage api
getModules	Provides the modules that belongs to the current application. You can decide if this request supports offline and also in which thread this request should be done
printModulesMetaData	This method is intented for development purposes and provides all modules meta-data information from network into the log
getServerVersion	Provides the server version for the current sdk. This tells if the sdk is outdated or not
requestToken	Requests a new authentication token
isAppAuthentication	Tells if the current authentication is based in the app credentials
isPasswordAuthentication	Tells if the current authentication is based in the password credentials
syncDevice	Syncs the current device with the device in HALO
subscribeForDeviceSync	Subscribes for the device update

Method name	Functionality
syncDeviceWhenNetworkAvailable	Synchronizes the device stored with the one in the server once the network is available. In this case callback is not allowed
sendDevice	Updates the device that is present in the core sending it to the server
getCurrentDevice	Provides the current cached device
addDeviceTag	Adds a new tag to the device to segment information and syncs it
addDeviceTags	Adds multiple tags to the device and syncs it
removeDeviceTag	Removes a tag from the device and syncs it
removeDeviceTags	Removes multiple tags
sendEvent	Send tracking analytic events of the current user.

Example: request the modules

Here is a full example on how to request the modules for the current app:

```
halo.core().manager()
    .getModules(Data.NETWORK_AND_STORAGE)
    .asContent()
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<List<HaloModule>>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<List<HaloModul
e>> result) {
                        //Manage the result
                }
        });
```

Example: request the modules metadata information

Here is a full example on how to request the modules metadata for the current app:

```
//print module metadata into log
halo.core().manager()
    .printModulesMetaData();
```

Example: Add a device tag

Here is the full example to add a tag to the current device and sync this device with HALO:

```
HaloSegmentationTag tagToAdd = new HaloSegmentationTag("myNewTagNam
e","myNewTagValue");
halo.core().manager()
    .addDeviceTag(tagToAdd)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<Device>() {
                @Override
            public void onFinish(@NonNull HaloResultV2<Device> resul
t) {
                        //Do something with the new device
                }
        });
```

Example: Send event

Here is the full example to send a analytic event to HALO:

```
//create custom object to send as an extra
HashMap<String, Object> eventValues = new HashMap<>();
eventValues.put("idUser", userId);
eventValues.put("userName",userName);
//create halo event to send
HaloEvent event = HaloEvent.builder()
        .withType(HaloEvent.REGISTER_LOCATION)
        .withLocation(locationAsString) //locationAsString = "lat,lo
ng"
        .withExtra(eventValues)
        .build();
//send event
halo.core().manager()
        .sendEvent(event)
        .execute(new CallbackV2<HaloEvent>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<HaloEvent> re
sult) {
                //do something with the event
            }
        });
```

Android SDK - Cloudinary Media Helper

HALO uses the Cloudinary service to upload media files. We have added to our SDK a helper with the filtering options. Check the javadoc documentation to learn more about this helper.

Here we are providing an example to blur an image provided:

```
HaloCloudinary.builder()
                .addEffects(HaloCloudinary.Effects.blur(100))
                .build("http://cloudinary.com/..."); //Cloudinary image url
```

Android SDK - Custom Middleware Requests

It is likely that you want to create a custom module and add its functionalities to HALO. You can do it using HALO by adding his module endpoint into the administration console. Use the help on the CMS to make sure you configure it properly.

When a new module is added, it generates urls in the following way:

```
https://halo.mobgen.com/api/---proxy---/-/
```

Keep in mind that the endpoint depends on the environment configured when Halo is installed and the default environment is production.

Using the SDK you can make custom requests to your endpoint to get the data, already being authenticated using the HALO authentication system. The available API to do so includes using the `HaloMiddlewareRequest` class. If, for example, we would like to make a request to our custom module with:

1. The company name “mobgen”
2. The Module type “cake”
3. The final url being `cakes/{id}`
4. Is under a proxy

The final url result would be:

```
https://halo.mobgen.com/api/---proxy---/mobgen-cake/cakes/{id}
```

Where id is the id of the cake you want to get. In the following pieze of code you will get an example of how to bring this data from the web service. Remember you can only make this request if HALO is already installed.

```
Map<String, String> params = new HashMap<String,String>(1);
params.put("id", myId);
HaloMiddlewareRequest.builder(halo.framework().network().client())
                  .method(HaloRequestMethod.GET)
                  .module("mobgen", "cake")
                  .hasProxy(true)
                  .url("cakes/{id}", params)
                  .callback(callback)
                  .build().execute();
```

The response that comes from this web service call is an okHttp3 `Response` object, so you can rely on their documentation to process it.

Android SDK - Offline support

The SDK supports an offline database that caches the responses for many requests and provides some cached data when the device is offline or cannot stablish a connection.

Most of the requests support a flag to tell the request how it should be synchronized with the server and which kind of that should provide. This flags belong to the `Data` class and can take the following values:

- **NETWORK_AND_STORAGE**: Will make the request and cache the response, providing this cached response. If the device has no internet connection will provide the previously cached response, or nothing if there is no data inside.
- **NETWORK_ONLY**: Will make the request to the network and provide the same data received bypassing the cache.
- **STORAGE_ONLY**: Will get the cached data no matter if there is internet connection or not in the device.

The result of a request using the sync engine comes in a `HaloResultV2` which contains two methods:

- **status()**: Provides the status of the data. It allows you to know if there was any error while performing the operation and the status of the thata (wether it is fresh, local or inconsistent).
- **data()**: Provides the data brought. It can be null if there was an error or no data is cached while in offline mode. You must always nullcheck it.

The callback for an operation comes only with one method which is `onFinish` and receives a `HaloResultV2` with the status and the data.

Example

```
halo.core().manager().getModules(Data.NETWORK_AND_STORAGE)
        .asContent()
        .threading(Threading.POOL_QUEUE_POLICY)
        .execute(new CallbackV2<List<HaloModule>>() {
          @Override
          public void onFinish(@NonNull HaloResultV2<List<HaloRemote
Module>> result) {
              if(result.status().isSuccess()){
                  showData(result)
              }else{
                  showError(result)
              }
          }
        });
```

Change Server Environment and Proxy Debugging

Change server environment

Maybe you don't want to use HALO in the default production environment (https://halo.mobgen.com (https://halo.mobgen.com)). To change the environment in which HALO is working on, you have to customize your installation process. See the example above:

```
Halo.installer(context)
        .environment("https://halo-int.mobgen.com")
        .install();
```

Disable SSL pinning

Maybe you don't want to use the SSL pinning in the HALO SDK that is enabled by default. To disable pinning you have to customize your installation process. This is not recomended but could be interesting for development purposes. See the example above:

```
Halo.installer(context)
        .environment("https://halo-int.mobgen.com")
        .disablePinning()
        .install();
```

Proxy debugging

You must follow the next steps to debug all requests information with some proxy debugger like Charles.

Install the certificate

You will need to install a certificate in your device to allow the proxy debugger application. If you are using Charles you can just download it from Help > Certificate.

Trust on this certificate

You need to add configuration to your app in order to have it trust the SSL certificates generated by SSL proxying debugger. This means that you can only use SSL Proxying with apps that you control. You must override the network security configuration (https://developer.android.com/training/articles/security-config.html) as follows:

```
<network-security-config>
  <debug-overrides>
    <trust-anchors>
      <!-- Trust user added CAs while debuggable only -->
      <certificates src="user" />
    </trust-anchors>
  </debug-overrides>
</network-security-config>
```

```
<applicationandroid:networkSecurityConfig="@xml/network_security_con
fig" ... >
        ...
</application>
```

Disable SSL pinning

You must disable SSL pinning on the HALO installer as follows:

```
Halo.installer(context)
        .environment("https://halo-int.mobgen.com")
        .disablePinning()
        .install();
```

Android SDK - Enable Debug Mode

Debug mode

HALO supports a debugging mode that allows you to see many information of what is going on behind the scenes. To enable this mode you have to add it in the installer instance.

```
Halo.installer(context)
        .debug(true)
        .install();
```

This change will:

1. Enable the logging for HALO.
2. Take the client id debug and the client secret debug configuration.
3. Take the debug id to enable the push notifications if it is available.

Print log information to file

HALO can also print the log information to file if debug mode is enabled. You have to add it in the installer instance as follows:

```
Halo.installer(context)
        .debug(true)
        .printLogToFile(PrintLog.SINGLE_FILE_POLICY)
        .install();
```

Print log policies

You can choose between three different policies to print the log to file:

Policy	
`NO_FILE_POLICY`	do not print any log (default behaviour)
`SINGLE_FILE_POLICY`	print each app execution in a single log file
`MULTIPLE_FILE_POLICY`	print each execution in a new log file

Android SDK - Content SDK Overview

Download 2.5.2 (https://bintray.com/halo-mobgen/maven/HALO-Content/_latestVersion)

Add dependency

In the HALO plugin add the following to enable the content sdk.

```
apply plugin: 'halo'

halo {
        ...
        services {
                content true
        }
        ...
}
```

Content API methods

The content API is the facade for the Content HALO SDK. Importing this library will need a valid HALO instance configured with some credentials. The HALO Content SDK allows the user to retrieve instances from the HALO Backend in two main ways:

- Search
- Sync

☑ **Tip:** Use the search method to get conent when you want to get some concrete elements, segmented data or certain information.

☑ **Tip:** Use the sync method if you prefer to download the whole module to use if offline. It is better for performance than search.

Creating an instance of the Content API is really simple once you have your HALO running. Just write the following line:

```
HaloContentApi contentApi = HaloContentApi.with(halo);
```

In addition you can provide a default locale to include it in all the queries even if you didn't provide it in the search/sync.

Search

If you want to bring certain data based on some criteria or search some of them in the HALO Backed, you have to use the search operation in the API. To do that we provide a class called `SearchQuery` with a simple fluent API that allows you to specify the criterias for the search. Once selected, you can perform the search and cache the result in local for offline use. See the example above:

```
SearchQuery query = SearchQuery.builder()
        .moduleIds("myModuleId")
        .beginSearch()
                .eq("name", "Sample")
        .end()
        .build();
contentApi.search(Data.NETWORK_ONLY, query)
        .asContent()
        .execute(callback);
```

This search will request all the instances for the module id “myModuleId” and which body contains a name with the value “Sample”. Check out the rest of the available options in the detailed documentation (page 37).

❗ **Warning:** The query builder search will return all results available including deleted or draft items. You must provide the appropiate query to return only published ones.

SearchQuery Factory

If you want to use common search options the `SearchQueryBuilderFactory` helps you to get items that has been published or removed or drafted.

```
SearchQuery query = SearchQueryBuilderFactory.getPublishedItemsByNam
e(moduleName, searchTag, searchQuery)
    .onePage(true)
    .segmentWithDevice()
    .build();
contentApi.search(Data.NETWORK_ONLY, query)
    .asContent()
    .execute(callback);
```

Here you can find the list of predefined search common operations with the `SearchQueryBuilderFactory` :

SearchQueryBuilderFactory	Explanation
getPublishedItems	Brings all the published items for the given module.
getPublishedItemsByName	Brings all the published items for the given module filtered by name.
getItemsByContentValue	Brings all items for the given module filtered by a content value.
getExpiredItems	Provides the expired items.
getArchivedItems	Provides the archived items.
getLastUpdatedItems	The updated items during the given millis from now.
getDraftItems	Provides the draft items for the given module.

Sync

The sync operation is thought for performance critical tasks. It allows to synchronize a full module consuming the less amount of data possible.

Syncing involves three steps:

- Listening for sync updates.
- Requesting a sync for a module.
- Requesting synced local instances.

Check out the following example for a full sync lifecycle:

Listen for sync updates

```
//Start listening for updates
ISubscription subscription = contentApi.subscribeToSync("my module n
ame", listener);
//When you are done or to free memory
subscription.unsubscribe();
```

Request a sync for a module

```
//Create the sync query
SyncQuery query = SyncQuery.create("my module name", Threading.POO
L_QUEUE_POLICY);
contentApi.sync(query);
```

Request the synced instances

```
//Request the synced content
contentApi.getSyncInstances("my module name")
        .asContent(MyCustomClass.class)
        .execute(callback);
```

Note: Make sure your MyCustomClass.class is properly annotated with LoganSquare `@JsonObject` annotation to make it work properly, otherwise the result will not be parsed. You can check it in content parsing section (page 44).

If you want to go in deep into this module, please refer to the detailed documentation (page 37).

Edit Content API

Basic content manipulation

The Edit Content API is the way to manipulate the general content instances. If you have the proper credentials you will be able to create, update or delete general content instances. See Halo Auth API (page 81) to get apropiate credentials.

For example if you want to update an instance.

Important: Please refer to the detailed documentation (page 46) to see other operations.

You must set a map with the content values. In this example: title and backgroundColor)

```
Map<String,Object> values = new HashMap<>();
values.put("title","the title");
values.put("backgroundColor", "#987654");
```

or set a custom object properly configured

Note: Make sure your MyCustomClass.class is properly annotated with LoganSquare `@JsonObject` annotation to make it work properly, otherwise the result will not be parsed. You can check it in content parsing section (page 44).

```
MyCustomClass values = new MyCustomClass("the title","#987654");
```

You must provide the item id, module id, instance name. As optional parameters you should provide segmentation tags, publication date and deletion date:

```
HaloContentInstance.Builder instanceBuilder = new HaloContentInstanc
e.Builder(moduleName)
    .withId(instanceId)
    .withModuleId(moduleId)
    .withPublishDate(publishDate)
    .withName(instanceName)
    .withContentData(values);
```

```
HaloContentEditApi.with(halo)
    .updateContent(instanceBuilder.build())
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<HaloContentInstance>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<HaloContentInstan
ce> result) {
            if(result.status().isSecurityError()){
               //there is an authentication error. Notify user to lo
gin.
            } else {
                if(result.data()!=null) {
                    //handle result of the update operatio
n.
                }
            }
        }
     });
```

If you want to go in deep into this module, please refer to the detailed documentation (page 46).

Advanced content manipulation

If you want to be able to add, modify and remove content instances in advanced use cases you must use the batch operation. You need to provide a `BatchOperations` which contains all the operations to perfom.

```
BatchOperations operations = new BatchOperations.Builder()
    .create(createInstances)
    .delete(deleteInstances)
    .update(updateInstaces)
    .truncate(truncateInstance)
    .build();
HaloContentEditApi.with(halo
    .batch(operations, true)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<BatchOperationResults>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<BatchOperationRes
ults> result) {
            //handle response with halo backend results
        }
    });
```

If you want to go in deep into this module, please refer to the detailed documentation (page 51).

Code generation tool to annotate custom models

The Content SDK provides an API to generate code based on annotations. The HALO plugin will fetch all necessary dependencies to generate code using Java Poet in the `AbstractProcessor`.

⚠ **Important:** Please refer to Java Poet (https://github.com/square/javapoet) to go in deep about code generation.

You can use this feature if you would like to:

- use the code generation tool in compile time to make sure that the proper code is generated to be able to create tables based on my custom model content structurethe.
- use the code generation tool to create a shared table with the version of

the table based on your custom model.

- use the generated code to be able to perform queries into the local database based on your custom model content (insert, delete, update or select) in the same way as we use the general content API.

Check out the following example creating a query to select in a database table by title.

Perfom a select query on a custom table with the generation tool

First create the generated database using the `HalocontentApi` with a new instance of `GeneratedDatabaseFromModel` that was autogenerated in compile time. This operation should be done in the Application to ensure the database was created after launching the app.

```
//...
HaloContentApi.with(haloinstance, locale, new GeneratedDatabaseFromM
odel());
//...
```

Then annotate your model with the `@HaloSearchableAnnotation` to create the database table and with the `@HaloQuery` annotation you will generate the code to perfom the query with the `HaloContentQueryApi` .

Note: The `@HaloQuery` parameters must be declared with the following format **@{name:class}**

Note: The `@HaloQueries` annotation is just an array of `@HaloQuery` annotations

```
//...
@HaloQueries(queries = {@HaloQuery(name="selectByTitle",query="selec
t * from Article where Title = @{mTitle:String}")})
@HaloSearchable(version = 13 , tableName = "Article")
public class Article implements Parcelable {
//...
```

Then annotate your model constructor with the `@HaloConstructor` annotation to indicate the names of the columns of your autogenerated table.

Warning: The order of the attributes must be the same on the annotation and in the constructor.

```
//...
@HaloConstructor(columnNames = {"Title","Date","ContentHtml","Summar
y","Thumbnail","Image"})
public Article(String title, Date date, String article, String summa
ry, String thumnail, String image) {
    mTitle = title;
    mDate = date;
    mArticle = article;
    mSummary = summary;
    mThumbnail = thumnail;
    mImage = image;
}
//...
```

This annotations will generate the code on the **build/generated/source/apt/** folder. You only need to import `com.mobgen.halo.android.app.generated` package to use the `HaloContentQueryApi` to perfom the operation as follows.

☑ **Tip:** The way to use the autogenerated `HaloContentQueryApi` is the same as in the general content API.

```
//...
HaloContentQueryApi.with(haloInstance).selectByTitle("My article tit
le.")
    .asContent(Article.class)
    .execute(new CallbackV2<List<Article>>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<List<Article>> re
sult) {
            if(result.data().size()!=0){
                populateArticles();
            }
        }
    });
//...
```

If you want to go in deep into this module, please refer to the detailed documentation (page 53).

Android SDK - Content SDK Detailed APIs

Here you can find fine grained explanations for every public param of the content SDK. The rest of the library is obfuscated over proguard and only intended methods are public and properly named although the code is (and will be) Open Source.

Search

With the search query you can request some instances from the HALO Backend based on some query parameters. See the Search query section for all the available params.

```
HaloContentApi api = HaloContentApi.with(halo);
api.search(Data.NETWORK_AND_STORAGE, query)
        .asContent()
        .execute(callback);
```

Search Query

The `SearchQuery` object supports many params to help in the search task. To create a new instance of the `SearchQuery` you have to use the `Builder` pattern by calling `SearchQuery.builder()`. The build object is parcelable and so you can send it across activities if needed.

Here you can find the full list of options you can chain into the `SearchQuery`:

Search param	Explanation
moduleName	requests a module by its name. You must owe it and it must be available for client applications.
moduleIds	the list of module ids to request.
instanceIds	the list of instance ids to request.
addRelatedInstances	add single relationship to filter the request
relatedInstances	the list of relationships to filter the request
allRelatedInstances	Request all relationships related to fieldname given

Search param	Explanation
pickFields	filters the values returned from the api, so it will not send more information than the needed by the application.
segmentMode	specifies how the segmentation should work against the tags. It can take two values: `PARTIAL_MATCH` or `TOTAL_MATCH`. Partial match checks if there is at least one tag in the content provided while total match ensures the content retrieved contains all the tags provided.
tags	segmentation tags to apply to this content. This param is related to the segmentMode one since it selects and segments the content based on those tags.
setDevice	applies all the tags from the current device and matches the segment mode to `PARTIAL_MATCH` if no mode was set. Refer to the segmentMode param.
segmentWithDevice	applies automatically the current device in the core to the search specified.
populateAll	if there are fields with relations, they are populated.
populate	a list of the fields that should be populated.
beginSearch/end	syntax declaration to make custom queries inside the content info. For example, if we have instances in halo with two fields, name and amount, we would be able to filter the searched instances based in both values.
beginMetaSearch/end	it has the same concept as the search but allows the user to filter based on metadata of the instance, such as the updated time or the instance name.
searchTag	tag name that will be used as an id for offline caching. This allows to override previous search data tagging them. It is useful for searches that include timestamps, since those searches would generate much more content than the needed for offline.

Search param	Explanation
locale	the locale specified for localized fields. If no locale is provided an object with all the locales will be provided for the given field.
ttl	time that the content should remain available offline.
pagination	indicates which page and which limit should be requested to get the instances.
onePage	allows to make a request with a single page. It is equivalent to make the request without pagination but provides the information as if you did it in a single page. The priority of this param is higher than the pagination one.
sort	allows to order by [asc or desc] any metatada field of the instance (name, publishedAt, createdAt, archivedAt, removedAt, deletedAt).
serverCache	set a time in seconds to cache the server response.

In the beginSearch/end and beginMetaSearch/end parameters there are many query parameters supported. Here you have an index on how to use them and an example.

Search condition	Explanation
and	Adds a condition to make both expression work together with an and condition.
or	Adds a contition to make both expressions work together with an or condition.
in	Check if the field has the values provided inside it.
nin	Ensures the field has not the values inside.
eq	Checks fields that are equals to the given value.
neq	Checks the fields are not equals to the given value.
lt	Checks the value is less than the provided value.

Search condition	Explanation
lte	Checks the value is less than or equals the provided value.
gt	Checks the value is greater than the provided value.
gte	Checks the value is greater than or equals the provided value.
like	Search a value that match with the provided value (you must provide at least 3 characters).
beginGroup	Begins a parenthesis group.
endGroup	Ends a parenthesis group.

Custom search sample

```
SearchQuery.builder()
    .beginSearch()
        .in("name", listOfNames)
        .and()
        .beginGroup()
            .nin("name", bannedNames)
            .and()
            .eq("active", "Activated")
        .endGroup()
    .end()
    .build();
```

If the expression built for the search is not correct it will throw a Runtime exception.

Data provider

The search supports 3 modes to select the source where the content comes from:

- **Data.NETWORK_ONLY**: this would be the simplest one. Just does the request and provides you the result.
- **Data.NETWORK_AND_STORAGE**: in this case the data is brought from network, stored in the local storage and retrieved to the user.
- **Data.STORAGE_ONLY**: this option provides only the cached data for

the given request.

Data parsing

When you call the `api.search` method you are not actually doing the request, it provides you an object that can be further configured to bring the data in the format you expect. In this case you have 3 different options:

- `asRaw()` : provides a cursor you can parse by your own. Usually this will not be used unless you need some sort of performance critical task in a list.
- `asContent()` : provides a `HaloContentInstance` list. This can be useful if you need to check also the metadata. To parse it to a custom class you can use the following operation on each instance:

```
HaloContentHelper.from(instance, MyCustomClass.class, halo.framewor
k().parser());
```

- `asContent(Class<T> clazz)` : this is the typical configuration you will need. Just pass your custom class to the content parameter and it will parse the list for you directly from the json received. Also the class must be annotated with `@JsonObject` and the fields with `@JsonField` . Refer to LoganSquare documentation (https://github.com/bluelinelabs/LoganSquare) for more details.

⚠ **Important:** Remember that to make the class available for parsing you need to use the correct annotations. See the content parsing section (page 44).

Sync

When a given module has so many items that handling them takes too much time or you want to have some content available in the background for the user, synchronization can make that work for you.

The process of the synchronization is the following:

- Subscribe to the synchronization hub to ensure when the synchronization process is done.

```
HaloContentApi contentApi = HaloContentApi.with(halo);
ISubscription subscription = contentApi.subscribeToSync(moduleName,
new HaloSyncListener() {
    @Override
    public void onSyncFinished(@NonNull HaloStatus status, @Nullabl
e HaloSyncLog log) {
            if(log != null){
                Log.i("Sync", log.toString());
        } else {
                Log.e("Sync", "Error " + status);
        }
    }
});
```

- Then you can request an internal module name to be synced:

```
SyncQuery query = SyncQuery.create("my module name", Threading.POO
L_QUEUE_POLICY);
contentApi.sync(query);
```

- You can then get the instances of the synced module even if you are offline:

```
contentApi.getSyncInstances("my module name")
        .asContent(MySampleClass.class)
        .execute(callback);
```

- Remember to unsubscribe from the `Subscription` created when you are done, this will avoid possible memory leaks:

```
subscription.unsubscribe();
```

- For debugging purposes, we keep a log for all the synchronizations that are done by an application. You can take them by module or all using the following call:

```
contentApi.getSyncLog("my module name")
        .asContent()
        .execute(callback);
```

- Finally you can remove your stored data for a given module using:

```
contentApi.clearSyncInstances("my module name")
        .asContent()
        .execute(callback);
```

The result provided by all the callbacks is a HaloResult as many other calls, so you can always take advantage of the state of the data and the possible errors that can appear, whether they are from the database or network.

Execution options

This api supports some execution options based on the sdk. Checkout them to see how productive you can be.

Response with callback in any thread: execute

Tip: You can provide Threading mode and get the result of the operation on the callback).

```
SearchQuery query = SearchQuery.builder()
    .moduleIds("myModuleId")
    .build();
contentApi.search(Data.NETWORK_ONLY, query)
    .asContent(MyModel.class)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<List<MyModel>>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<List<MyMode
l>> result) {
                //Handle result on the callback
            }
        );
```

Response inline in the same thread: excecuteInline

Tip: You can execute sync requests on same execution thread (**Threading.SAME_THREAD_POLICY**) that returns responses (inline, no callback needed).

Warning: The execution will be on the same thread so you can not modify Threading mode.

```
SearchQuery query = SearchQuery.builder()
     .moduleIds("myModuleId")
     .build();
HaloResultV2<List<MyModel>> response = contentApi.search(Data.NETWOR
K_ONLY, query)
     .asContent(MyModel.class)
     .executeInline();
```

Threading

Almost every request supports the threading mode. This mode allows you to select which is the threading context in which the request will be executed. Lets say you are already in another thread and you don't want to spawn another one, with this param you can specify this behavior. There are 3 modes supported:

- **Threading.POOL_QUEUE_POLICY**: spawns a new thread into a thread pool that will be executed as soon as possible.
- **Threading.SINGLE_QUEUE_POLICY**: spawns a new thread into a thread queue that will execute it once the other threads enqueued free the queue.
- **Threading.SAME_THREAD_POLICY**: does not spawn a thread, it uses the same context as it was called so everything will be executed synchronously.

Content parsing

We use LoganSquare for performance reasons inside our sdk. It allows us to parse really fast and without too much methods (avoiding the method count limit). Here we are providing a sample of how to annotate some content to parse it properly. Refer to LoganSquare documentation (https://github.com/bluelinelabs/LoganSquare) for more details.

Mapping sample

```
@JsonObject
public class Article {

    @JsonField(name = "Title")
    public String mTitle;

    @JsonField(name = "Date")
    public Date mDate;

    @JsonField(name = "ContentHtml")
    public String mArticle;

    @JsonField(name = "Summary")
    public String mSummary;

    @JsonField(name = "Thumbnail")
    public String mThumbnail;

    @JsonField(name = "Image")
    public String mImage;
}
```

Android SDK - Content SDK Detailed APIs

Here you can find fine grained explanations for every public param of the edit content SDK. The rest of the library is obfuscated over proguard and only intended methods are public and properly named although the code is (and will be) Open Source.

With the edit content API you can add, modify or delete general content instances if you have appropiate credentials. See Halo Auth API (page 81) to get apropiate credentials.

HaloContentInstance

To create, update or delete a general content instance on the server you must provide a valid `HaloContentInstance` . You have to use the fluent API to create a valid object through `Builder` pattern by calling `HaloContentInstance.builder()` .

The build object is parcelable and so you can send it across activities if needed.

```
HaloContentInstance.Builder instanceBuilder = new HaloContentInstanc
e.Builder(moduleName)
    .withId(instanceId)
    .withModuleId(moduleId)
    .withPublishDate(publishDate)
    .withName(instanceName)
    .withContentData(values);
```

Here you have the complete list of methods you can use to create a `HaloContentInstance.Builder` .

HaloContentInstance	Explanation
withId	the item id to use on update or delete operations.
withModuleId	the module id.
withAuthor	the instance author.
withName	the name of the content item.
withTags	the segmentation tags to the content instance.

HaloContentInstance	Explanation
withContentData	the content values of the instance.
withCreationDate	the creation date.
withLastUpdateDate	the last update date.
withPublishDate	the publish date.
withRemovalDate	the removal date shceduled.

The `withContentData` method will accept a Map of values or a properly annotated class.

Map values

Simply map the values.

```
Map<String,Object> values = new HashMap<>();
values.put("title","the title");
values.put("pubDate", "01/10");

HaloContentInstance.Builder instanceBuilder = new HaloContentInstanc
e.Builder(moduleName)
    .withId(instanceId)
    .withModuleId(moduleId)
    .withPublishDate(publishDate)
    .withName(instanceName)
    .withContentData(values);
```

Annotated class

Create your custom class with the fields properly annotated to map with the values.

```
@JsonObject
public class MyObjectAnnotated implements Parcelable {

    @JsonField(name = "title")
    String mTitle;

    @JsonField(name = "pubDate")
    Date mDate;

    ...
}
```

⚠ **Important:** Remember to annotate the class with the correct annotations from LoganSquare. Please refer to LoganSquare documentation (https://github.com/bluelinelabs/LoganSquare) for more details.

```
MyObjectAnnotated myObject = new MyObjectAnnotated(title, pubDate);

HaloContentInstance.Builder instanceBuilder = new HaloContentInstanc
e.Builder(moduleName)
    .withId(instanceId)
    .withModuleId(moduleId)
    .withPublishDate(publishDate)
    .withName(instanceName)
    .withContentData(myObject);
```

Finally to create the `HaloContentInstance` call the build method.

```
HaloContentInstance haloContentInstance = instanceBuilder.build();
```

Operations

☑ **Tip:** If you want to receive updates about sync process please remember to subscribe. The `HaloEditContentApi` perfoms internally a sync after every requested operation. Please refer to sync process documentation (page 41) to know more about sync.

```
ISubscription mSyncSubscription = HaloContentApi.with(halo).subscrib
eToSync(moduleName,this);
```

Add content

With the `addContent` method you can add new general content instances on the HALO Backend. To create a general content instance on the server you must provide a new valid `HaloContentInstance`. The result will provide feedback about any error (authentication, parsing or network exception) during the request or the result parsed as `HaloContentInstance`.

```
HaloContentEditApi.with(halo)
    .addContent(haloContentInstance)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<HaloContentInstance>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<HaloContentInstan
ce> result) {
            if(result.status().isSecurityError()){
               //there is an authentication error. Notify user to lo
gin.
            } else {
                if(result.data()!=null) {
                    //handle result of the add operatio
n.
                }
            }
        }
     });
```

Update Content

With the `updateContent` method you can update current general content instances on the HALO Backend. To update a general content instance on the server you must provide the `HaloContentInstance` to perfom the update. The result will provide feedback about any error (authentication, parsing or network exception) during the request or the result parsed as `HaloContentInstance`.

```
HaloContentEditApi.with(halo)
    .updateContent(haloContentInstance)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<HaloContentInstance>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<HaloContentInstan
ce> result) {
            if(result.status().isSecurityError()){
               //there is an authentication error. Notify user to lo
gin.
            } else {
                if(result.data()!=null) {
                    //handle result of the update operatio
n.
                }
            }
        }
     });
```

Delete Content

With the `updateContent` method you can remove current general content instances on the HALO Backend. To delete a general content instance on the server you must provide the `HaloContentInstance` to perfom the removal operation. The result will provide feedback about any error (authentication, parsing or network exception) during the request or the result parsed as `HaloContentInstance`.

```
HaloContentEditApi.with(halo)
    .deleteContent(haloContentInstance)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<HaloContentInstance>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<HaloContentInstan
ce> result) {
            if(result.status().isSecurityError()){
               //there is an authentication error. Notify user to lo
gin.
            } else {
                if(result.data()!=null) {
                    //handle result of the delete operatio
n.
                }
            }
        }
     });
```

Batch operations

When you need to perfom multiple operations like add, modify or remove content instances in advanced use cases you must use the batch operation method. You need to provide a `BatchOperations` which contains all the operations to perfom. The second parameter is to perfom or not a sync operation of the module.

Saving engine will handle two ways of saving changes:

- Directly by trying to make the call and failing if it is not possible giving a callback with `BatchOperationResults` which contains all opertions result ordered by operation type.
- In background so it is done when the internet connection is ready again by storing modifications in local temporarily. SDK user will be notified with a notification event if user was subscribed to receive this events.

```
BatchOperations operations = new BatchOperations.Builder()
    .create(createInstances)
    .delete(deleteInstances)
    .update(updateInstaces)
    .truncate(truncateInstance)
    .build();
HaloContentEditApi.with(halo
    .batch(operations, true)
    .threadPolicy(Threading.POOL_QUEUE_POLICY)
    .execute(new CallbackV2<BatchOperationResults>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<BatchOperationRes
ults> result) {
            //handle response with halo backend results
        }
    });
```

You should subscribe to batch events if you want to receive notifications with conflict operations (when server has a newer version of the instance) or to receive a notification event when the SDK tried to retry a previous batch operation that failed.

⚠ **Important:** Please remember to keep a reference to the subscrition.

```
ISubscription subscription = HaloContentEditApi.with(halo)
    .subscribeToBatch(new HaloContentEditApi.HaloBatchListener() {
        @Override
        public void onBatchConflict(@Nullable BatchOperations operat
ions) {
            //handle conflict operations as you need
        }

        @Override
        public void onBatchRetrySuccess(@NonNull HaloStatus status,
@Nullable BatchOperationResults operations) {
           //handle operations after internet connection works again
        }
    });
```

⚠ **Important:** Please remember to unsbscribe from events when your are done to avoid leaking

```
subscription.unsubscribe();
```

Android SDK - Code generation tool SDK

Code generation API

The HALO Content SDK provides some annotations to generate code on compilation time. When you annotate a custom model it will generate the following classes:

- **HaloTable$$ContentVersion**: provides a shared `HaloTable` to save the table custom model name and version. It will drop the table when the version has changed.
- **GeneratedDatabaseFromModel**: provides the code to generate the `HaloTable$$ContentVersion` table and a database table for each annotated model.
- **HaloTable$$ModelName**: provides the `HaloTable` with the name of your model to store information related to that model. It will generate one class per model annotated.
- **HaloContentQueryApi**: provides the API to perfom queries with the annotated models.

Note: For example, if you annotate a model called `Article` it will generate a `HaloTable$$Article` class.

Tip: The code generation tool will read supported annotations to generate the code and all classes will appear under **build/generated/source/apt/** folder on the `com.mobgen.halo.android.app.generated` package.

HALO annotations supported.

The SDK provide the following annotations to generate code on compilation time.

HALO annotations	Explanation
@HaloSearchable	Defines that a model can be used with the code generation tool.
@HaloConstructor	Provides the column names to the generator tool.

HALO annotations	Explanation
@HaloField	Define that a field from a class can be indexed on the local database.
@HaloQuery	Generates a class to perform custom queries.
@HaloQueries	Generates an array of HaloQueries.

Example of use

We will show you an example with a model called Article using all supported annotations. This will generate the classes and the `HaloContentQueryApi` to perfom the queries on the local database.

@HaloSearchable

You can use the `@HaloSearchable` annotation on the class declaration. It must receive two params:

- the version of the model
- the name of the table on the database.

```
//...
@HaloSearchable(version = 13 , tableName = "Article")
public class Article implements Parcelable {
//...
```

@HaloField

You can use the `@HaloField` annotation on a field of the class. You must provide the column name of the field to make an index.

```
//...
@HaloField(index = true,columnName = "Title")
String mTitle;
Date mDate;
String mArticle;
String mSummary;
String mThumbnail;
@HaloField(index = true,columnName = "Image")
String mImage;
...
```

@HaloConstructor

You can use the `@HaloConstructor` annotation on a constructor of the class. You must provide all the column names of the database table in the same order as in the constructor.

```
//...
@HaloConstructor(columnNames = {"Title","Date","ContentHtml","Summar
y","Thumbnail","Image"})
public Article(String title, Date date, String article, String summa
ry, String thumnail, String image) {
    mTitle = title;
    mDate = date;
    mArticle = article;
    mSummary = summary;
    mThumbnail = thumnail;
    mImage = image;
}
//...
```

@HaloQuery

You can use the `@HaloQuery` annotation on the class declaration. You must provide:

- a name for the method
- the query to perfom with the following format **@{nameOfParam:classOfParam}**

Warning: You must provide a `@HaloQueries` annotation array with all the `@HaloQuery` annotations you want to perfom.

Note: The `@HaloQuery` parameters must be declared with the following format **@{name:class}** as you see in the example.

```
//...
@HaloQueries(queries =
    {
        @HaloQuery(name="deleteByTitle", query=("delete from Articl
e where Title = @{mTitle:String}")),
        @HaloQuery(name="selectTitle",query="select * from Article w
here Title = @{mTitle:String}"),
        @HaloQuery(name="insertArticle",query="insert into Article(T
itle,Date,ContentHtml,Summary,Thumbnail,Image) VALUES (@{mTitle:Stri
ng},@{mDate:Date},@{mArticle:String},@{mSummary:String},@{mThumbnai
l:String},@{mImage:String});")
    })
public class Article implements Parcelable {
//...
```

HaloContentQueryApi

The abstract processor will generate a `HaloContentQueryApi` that works like the general content API with one method for each `@HaloQuery` annotation on any custom model. You can fetch the result of any query as raw cursor or as a custom content model using the `asContent()` method like in the general content API.

> **Warning:** Ensure you have created the generated database with `HaloContentApi.with(haloinstance, locale, new GeneratedDatabaseFro`
> before creating an instance of the `HaloContentQueryApi`

Using the HaloContentQueryApi

It is fairly recommendable to create the instance as a singleton in your application class or using Dagger after installing HALO. Creating an instance of the Content Query API is really simple once you have your HALO running. To create an instance of the `HalocontentQueryApi` just write this following lines:

```
HaloContentQueryApi haloContentQueryApiInstance = HaloContentQueryAp
i.with(haloInstance);
```

Each `@HaloQuery` annotation will generate a method to perfom the operation on the autogenerated class `HaloContentQueryApi` . To perfom the query from a model annotation with name `selectTitle` you must write the following code:

```
//Using my custom model Article.class to parse result
HaloContentQueryApi.with(haloInstance)
    .selectTitle("Search my title")
    .asContent(Article.class)
    .execute(new CallbackV2<List<Article>>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<List<Article>> re
sult) {
            if(result.data().size()!=0){
                insertArticles();
            }
        }
});;
```

Example of code generation tool usage

Here you can find a full example with the model called Article with HALO annotations and how to perfom queries to the database using the autogenerated `HaloContentQueryApi` .

Create the generated database

You must ensure that the autogenerated database was created in the Application with the following code:

Warning: The `GeneratedDatabaseFromModel` class should exist on build folder so if you are having troubles clean/rebuild your project.

```
//...
HaloContentApi.with(haloInstance, locale, new GeneratedDatabaseFromM
odel());
//...
```

Annotate your model

This is the custom model **Article.class** that we will use during the example.

```
//... imports
@HaloSearchable(version = 13 , tableName = "Article")
@HaloQueries(queries = {
        @HaloQuery(name="selectTitle",query="select * from Article w
here Title = @{mTitle:String}"),
        @HaloQuery(name="insertArticle",query="insert into Article(T
itle,Date,ContentHtml,Summary,Thumbnail,Image) VALUES (@{mTitle:Stri
ng},@{mDate:Date},@{mArticle:String},@{mSummary:String},@{mThumbnai
l:String},@{mImage:String});")
})
public class Article {
    @HaloField(index = true,columnName = "Title")
    String mTitle;
    Date mDate;
    String mArticle;
    String mSummary;
    String mThumbnail;
    @HaloField(index = true,columnName = "Image")
    String mImage;

    @HaloConstructor( columnNames = {"Title","Date","ContentHtml","S
ummary","Thumbnail","Image"})
    public Article(String title, Date date, String article, String s
ummary, String thumnail, String image) {
        mTitle = title;
        mDate = date;
        mArticle = article;
        mSummary = summary;
        mThumbnail = thumnail;
        mImage = image;
    }
}
```

Perfom the queries

Warning: The `HaloContentQuery` methods are autogenerated and the name of each one depends on the annotation you provide in your model.

Following the example to insert a new article into the database you will use the `insertArticle` method as follows:

```
HaloContentQueryApi.with(haloInstance).insertArticle("Title",new Dat
e(),"<p>Content</p>","Summary",null,mImage.getUrl())
    .asContent(Article.class)
    .execute(new CallbackV2<List<Article>>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<List<Article>> re
sult) {
            //use the result from the query
        }
    });
```

Following the example to select a article by title from the database you will use the `selectTitle` method as follows:

```
HaloContentQueryApi.with(haloInstance).selectTitle("This is a titl
e")
    .asContent(Article.class)
    .execute(new CallbackV2<List<Article>>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<List<Article>> re
sult) {
            //use the result from the query
        }
    });
```

Android SDK - Notifications SDK Overview

Download 2.5.2 (https://bintray.com/halo-mobgen/maven/HALO-Notifications/_latestVersion)

In the HALO plugin add the following to enable the notifications sdk.

```
apply plugin: 'halo'

halo {
        ...
        services {
                notifications true
        }
        ...
}
```

You also need to add the google play services json you download from the firebase console from google (`google-services.json`). Add it in a proper place to make this work.

Notifications API

To start using the notifications, in the same way you do with other sdks you must create a new instance using the already configured instance of HALO.

```
HaloNotificationsApi notificationsApi = HaloNotificationsApi.with(ha
lo);
```

❗ **Warning:** Since android O you have to create a notification channel to receive the push notification when your compileSdkVersion is 26 or greater

HALO will create a notification channel on devices with Android O but you can override that channel with your own channel with your own settings.

```
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_COD
ES.O) {
    NotificationChannel channel = new NotificationChannel("NOTIFICAT
ION_CHANNEL_ID",
            "CHANNEL_NAME", NotificationManager.IMPORTANCE_DEFAULT)
    channel.setShowBadge(false);
    notificationsApi.notificationChannel(channel);
}
```

Also you have to release the memory in the `onTerminate` method of your application:

```
notificationsApi.release();
```

Simple use

Listen all the notifications that arrive to the device by attaching a listener to the api.

```
ISubscription subscription = notificationsApi.listenAllNotification
s(listener);
```

Then you will receive a call on this callback everytime a notification is received. This will contain the bundle with data provided by the notifications service and a bundle with extra data if it is provided.

Once you are done with this listener you can cancel the subscription by calling `unsubscribe()`.

```
subscription.unsubscribe();
```

You can also customize your notifications by adding a notification decorator. You will receive as parameter the `NotificationCompat.Builder` already configured by HALO, so you can override its behavior. If a decorator returns `null` instead the `NotificationCompat.Builder` configured, the notification will not be displayed. Refer to the detailed documentation (page 0) to learn how to write your custom decorator.

```
notificationsApi.setNotificationDecorator(notificationDecorator);
```

Enable notifications usage

When a new notification is received on a device, the SDK will send a request to HALO reporting notification updates. There are three different events they have to report: receipt, open and dismiss. To enable this feature you must enable directly on the notification api singleton. Refer to the detailed documentation (page 0) to learn how to listen to notification events using the HALO SDK.

```
notificationsApi.enablePushEvents();
```

Android SDK - Notifications SDK Detailed APIs

Enable notifications

The notification system is built in in the SDK based on the FCM framework (Firebase Cloud Messages), which is the default system to receive notifications in the Android OS.

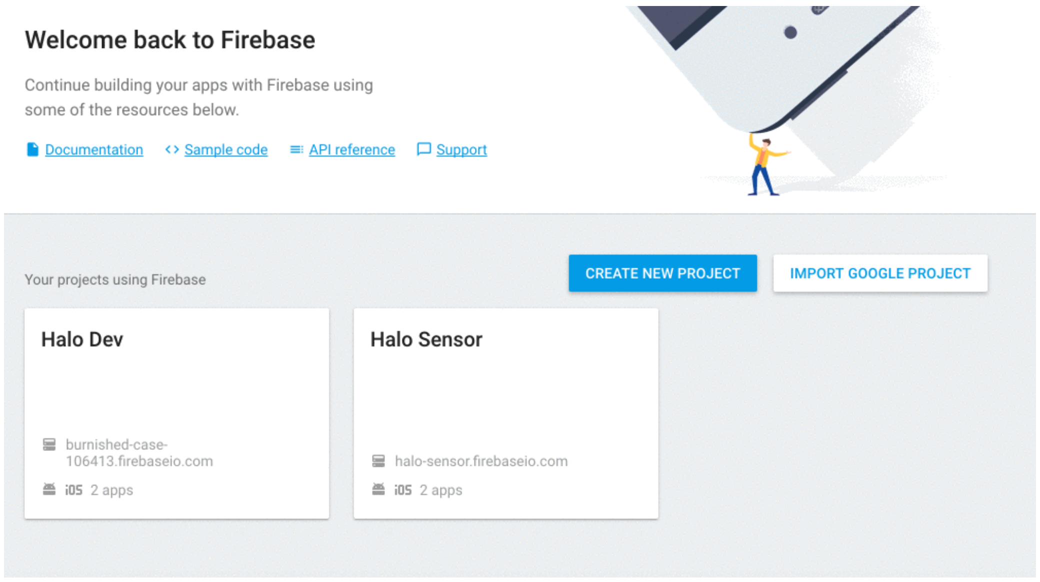
Warning: Since FCM uses the Google Play Services framework, HALO cannot receive notifications in those mobile phones that does not support the Google Play Store with Play services.

Based on the HALO backend you can send many data in the notifications and segment depending on the segmentation of the users present in the system. In this guide we will show you how to enable the notifications on the SDK and how to handle some custom actions on those notifications.

Step 1. Configure a FCM project

First of all we need to create a FCM project in the Firebase Console (https://console.developers.google.com). If you already have an FCM project configured you can go directly to 3. If you already have an app with the given package configured go to 6.

1. Click on 'Create New Project' button.
2. Fill in the project name and the country/region.
3. Enter the project and click on 'Add app'.
4. Click on 'Add Firebase to your Android App'.
5. Fill in your package name of the app and the debug certificate SHA-1.
6. Click on the app menu you want receive the push notifications and select 'Manage'.
7. There select cloud messaging.
8. Write down the server key. It looks like 'AIzaSyAtN64Y0****-*******'


Welcome back to Firebase
Continue building your apps with Firebase using some of the resources below.
Documentation
Sample code
API reference
Support
Your projects using Firebase
CREATE NEW PROJECT
IMPORT GOOGLE PROJECT
Halo Dev
burnished-case-106413.firebaseio.com
iOS 2 apps
Halo Sensor
halo-sensor.firebaseio.com
iOS 2 apps

Step 2. Add the Server key to HALO

Take the Server API Key obtained in the previous step and put it in the administration console of HALO.:

1. Do login in HALO as an admin.
2. Go to your apps.
3. Select the application you are enabling the push notifications to.
4. Update the app filling the **Android key** field with your server key.

Step 3. Enable the notifications in the SDK

To enable the notifications inside your app you have to add the HALO plugin (page 0). Once done, in your **HALO properties closure** enable the notifications service. Take the following code of a build.gradle as an example.

```
halo {
    ...
    services {
        notifications true
    }
    ...
}
```

Step 4. Add Google Play Configuration File

Take the file obtained in the Step 1 of this tutorial and place it in the root of your application module or in your flavor folder named as “*google-services.json*”.

React to a notification

In HALO we support two different notification types. **Normal notifications** and **silent notifications**. Normal notifications includes the UI that shows the user a notification was received, while silent notifications only notify you in a callback to perform some background work.

With this silent notifications the SDK will not display any UI but also will not perform any action. As a developer you have to put an entry point so we can send you the information received. In this guide we tell you how to do it. Follow the instructions below to add a listener.

1. Create the listener

This instance will receive the notifications for those notifications that will be listened with the `HaloNotificationListener`. Here you can check an example:

```
public class NotificationReceiver implements HaloNotificationListene
r {

    @Override
    public void onNotificationReceived(@NonNull Context context, @No
nNull String from, @NonNull Bundle data, @Nullable Bundle extra){
        //Do something with this data
    }
}
```

2. Attach this listener to HALO

You can attach the listener to listen **not silent**, **silent** or **all** notifications:

```
notificationsApi.listenAllNotifications(new NotificationReceive
r()); // All
notificationsApi.listenNotSilentNotifications(new NotificationReceiv
er()); //Not silent
notificationsApi.listenSilentNotifications(new NotificationReceive
r()); //Silent
```

Set custom notification id

You can attach custom id generation to change the way the notification will be created. Also you can modify the bundle data of the notification.

```
notificationsApi.customIdGeneration(new CustomIdGeneration() {
            @Override
            public int getNextNotificationId(@NonNull Bundle data, i
nt currentId) {
                //you can update your bundle
                data.putInt("customData","customData");
                //you can modify how notification id creation
                int miCustomID = generateNotificationIdMethod();
                return miCustomID;
            }
        });
```

Get UI notification id

When a notification is displayed in the UI, this notifications has an unique id assigned so you can perform some actions on it, like cancelling. You can access to this id in the data bundle provided in the notification callback. There is a helper method in the api to do so:

```
Integer notificationId = notificationsApi.getNotificationId(bundle);
```

Keep in mind this method returns null if there is no notification id attcached to the notification. This means there is no UI displayed linked to the received notification.

Customize notification displayed

Behind the scenes, when the server sends the notification to the Google Firebase Services, it can send information in two ways:

- **Providing a notification payload**: This generates a notification automatically and you cannot handle this behaviour.
- **Providing a data payload**: The data payload allows you to handle whatever action you need.

In Android, HALO always handles the notifications using the `NotificationService` and the data payload.

Since we create the notifications manually, we have designed every single functional addition to the notifications as a decorator pattern (https://en.wikipedia.org/wiki/Decorator_pattern). The base decorator class is `HaloNotificationDecorator` which is extended by many classes in the SDK that finally are chained to add the behaviour to the Android `NotificationCompat.Builder`.

To modify something in the notifications, we provide a method so you can handle or add behaviour to this notification just by extending the `HaloNotificationDecorator` class. Here you have the example guide to put an icon on the notification.

If you return `null` in your custom decorator the notification will not be displayed.

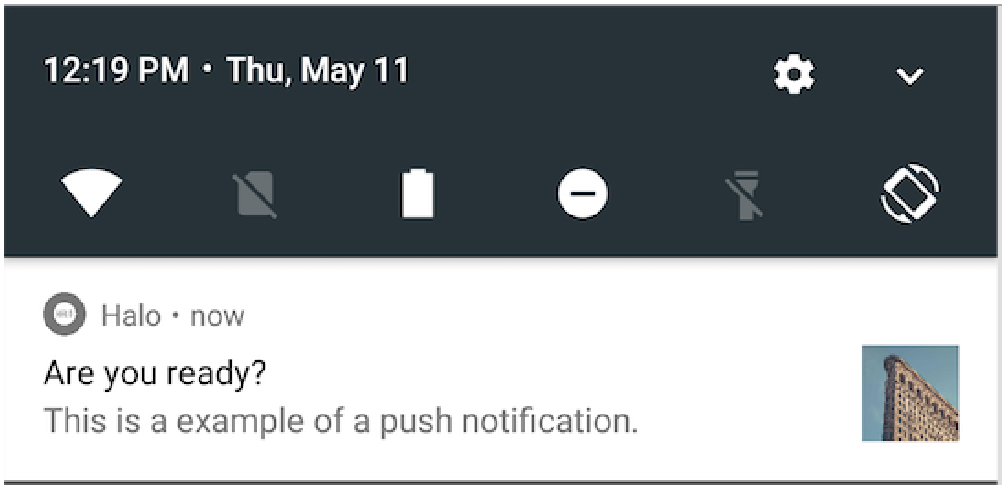
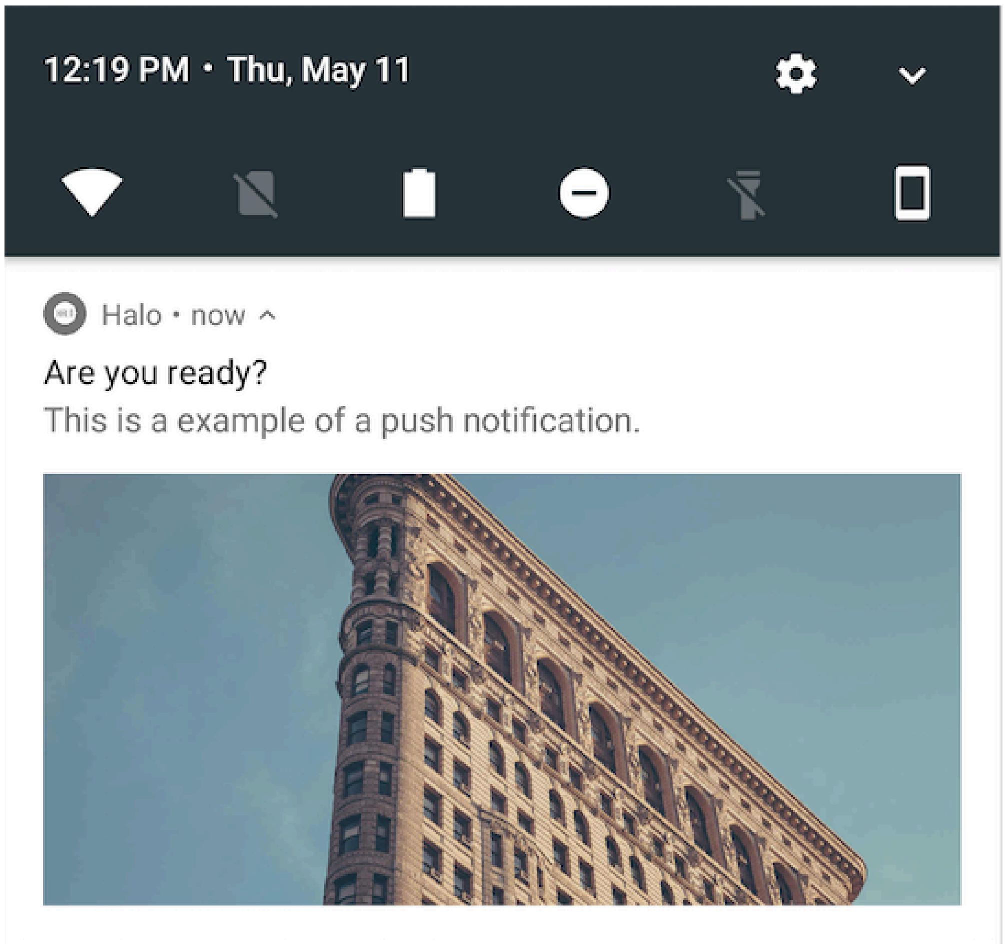
Notification with image support

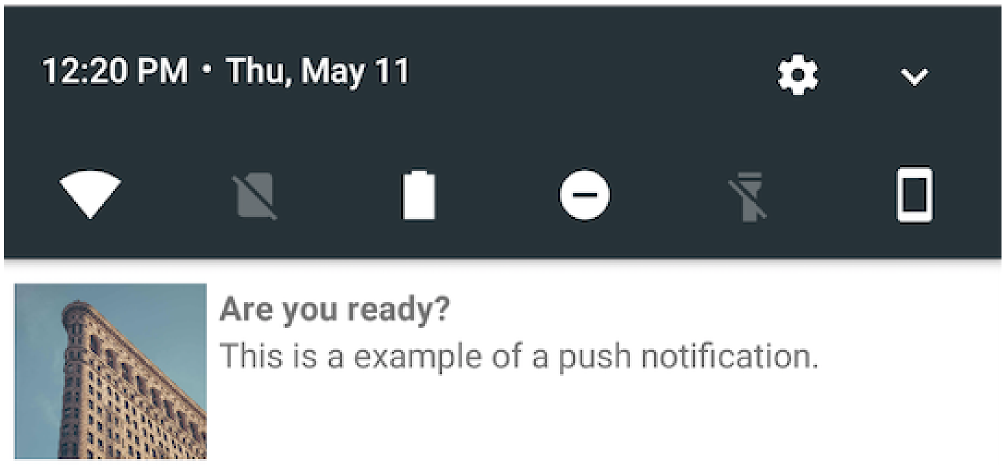
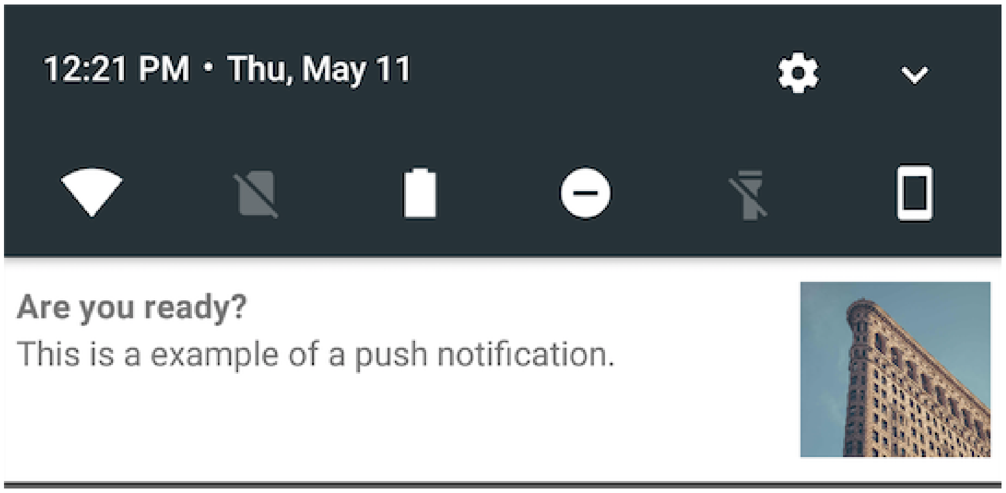
If your payload contains image object, as follows, HALO will handle different custom notification UI for you. The layout types are one of the following:

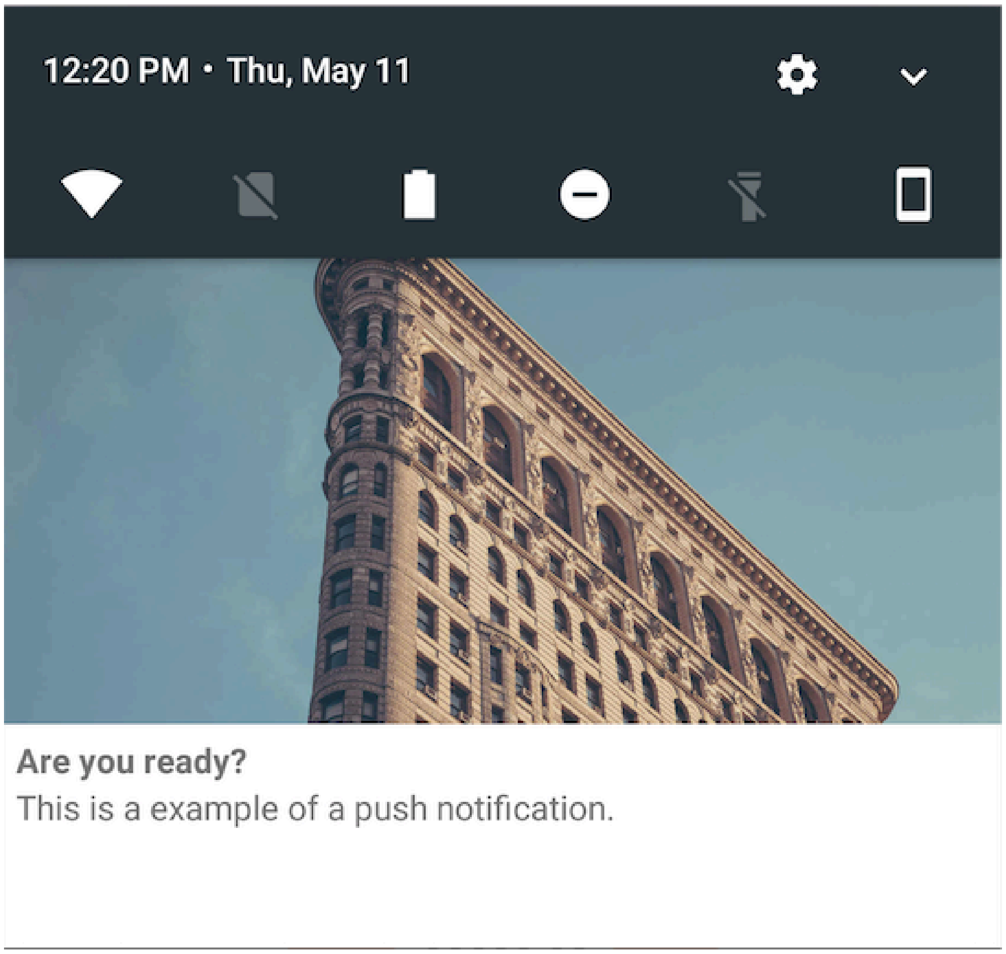
```
["default","top", "left", "right", "bottom", "background", "expande
d"]
```

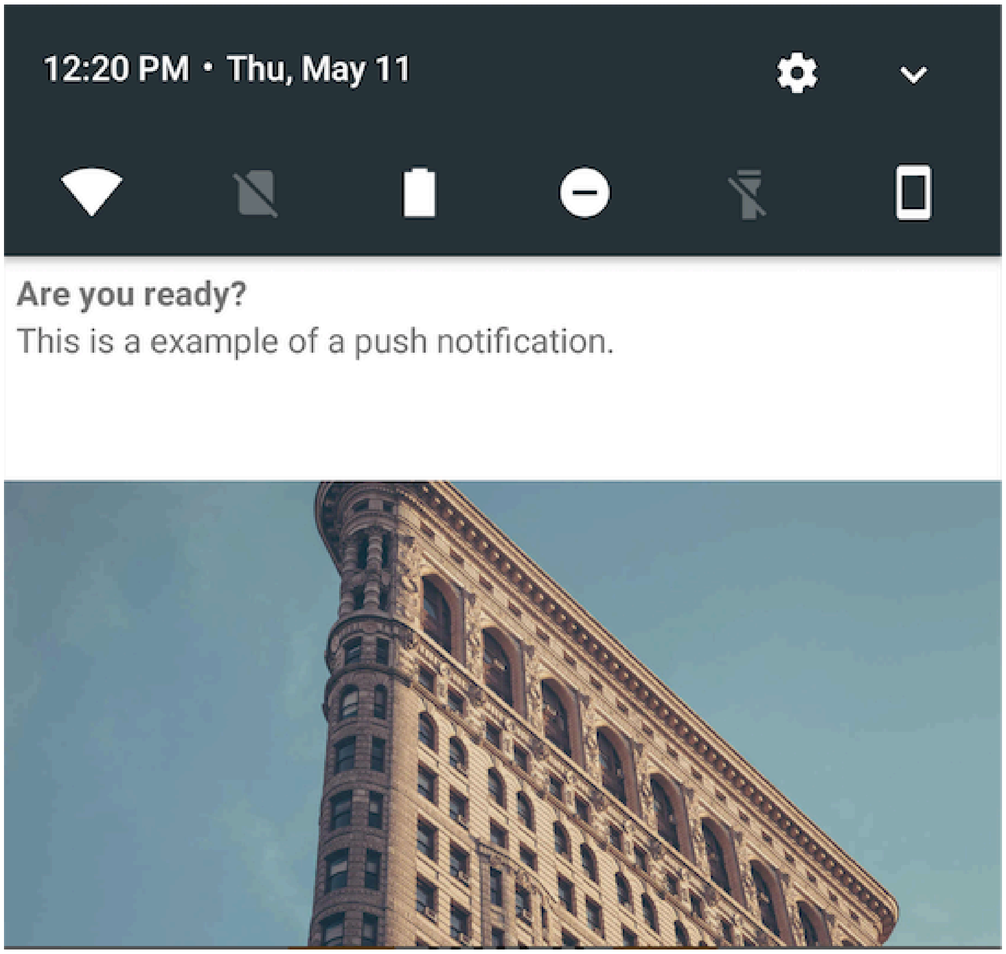
```
{
    "data": {
        "image": {
            "url": "http://imageurl" ,
            "layout": "background"
            }
      }
}
```

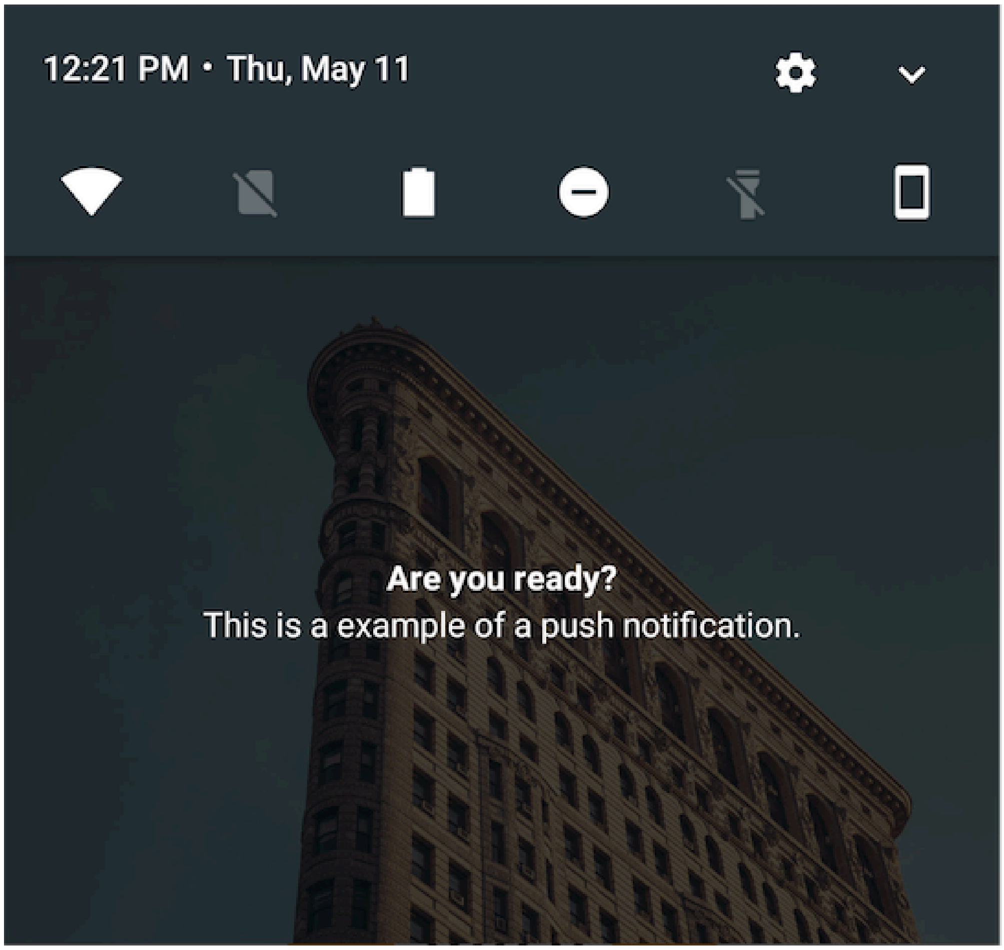
We show you the different layout configurations:

Layout type	Example
default	 12:19 PM • Thu, May 11 Halo • now Are you ready? This is a example of a push notification.
expanded	 12:19 PM • Thu, May 11 Halo • now Are you ready? This is a example of a push notification.

Layout type	**Example**
left	 **Are you ready?** This is a example of a push notification.
right	 **Are you ready?** This is a example of a push notification.

Layout type	Example
top	 12:20 PM • Thu, May 11 **Are you ready?** This is a example of a push notification.

Layout type	Example
bottom	 12:20 PM • Thu, May 11 **Are you ready?** This is a example of a push notification.

Layout type	Example
background	 Are you ready? This is a example of a push notification.

1. Create custom decorator

We are providing a custom implementation of the icon decorator:

```
public class CustomIconDecorator extends HaloNotificationDecorator {

    public NotificationIconDecorator(){
        super();
    }

    @Override
    public NotificationCompat.Builder decorate (@NonNull Notificatio
nCompat.Builder builder, @NonNull Bundle bundle) {
        builder.setSmallIcon(R.drawable.myNotificationIcon);
        //You can return also null if you want this notification no
t to appear
        return builder;
    }
}
```

2. Add decorator to HALO

If you want to add this decorator to the notification service you can do it while in the install process of HALO:

```
notificationsApi.setNotificationDecorator(new CustomIconDecorato
r());
```

As it is a Decorator, you can always chain as many decorators as you want by calling the method `NotificationDecorator#chain(builder, bundle)`.

This also can be used for example to execute an intent when the user clicks on the notification, and you are allowed to chain multiple decorators in the following way:

```
new PendingIntentDecorator(context, new CustomIconDecorator());
```

Example of a decorator to add notification behavior

```
public class PendingIntentDecorator extends HaloNotificationDecorato
r {

    private Context mContext;

    public CustomNotificationDecorator(Context context, HaloNotifica
tionDecorator decorator) {
        super(decorator);
        mContext = context;
    }

    @Override
    public NotificationCompat.Builder decorate(NotificationCompat.Bu
ilder builder, Bundle bundle) {
        Intent intent = new Intent(ctx, MainActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(ct
x, 0, intent, 0);
        builder.setContentIntent(pendingIntent);
        return chain(builder, bundle);
    }
}
```

If you want to get the custom decorator of the notification service you can do it:

```
notificationsApi.getNotificationDecorator();
```

Android SDK - Notifications SDK report event action APIs

Enable notifications usage

There are three different events they have to report: receipt, open and dismiss. To enable this feature you must enable directly on the notification api singleton. As a developer you can set a listener to be notified when a push action (receipt, open or dismiss) was reported to HALO.

Without any listener:

```
notificationsApi.enablePushEvents();
```

With a listener:

```
notificationsApi.enablePushEvents(new HaloNotificationEventListene
r() {
    @Override
    public void onEventReceived(@Nullable HaloPushEvent haloPushEven
t) {
        if (haloPushEvent != null) {
            Toast.makeText(halo.context(), "Action: " + haloPushEven
t.getAction(), Toast.LENGTH_SHORT).show();
        }
    }
});
```

Android SDK - Translations SDK Overview

Download 2.5.2 (https://bintray.com/halo-mobgen/maven/HALO-Translations/_latestVersion)

What is it for?

The HALO Translations SDK is a helper library that works as a plugin for HALO that, given a module name and the fields for the key and value it is able to bring the texts to your UI. The typical use case for this library goes over showing in the UI the translated texts that are present in the HALO backend. It makes it easy to make it work offline and asynchronous.

This library only supports one text for the same module at the same time so, once you change the language it will be cached removing the previous one.

Texts are synchronized based on the changes of the CMS, so if one text is removed, added or updated only those changes will be downloaded when the plugin is used. You can use push notifications also to redownload the texts if they have changed and you want them to be updated on the opened apps, but this use case is just a possible implementation and does not come directly implemented in the library.

Setup the lib

You can setup the translations using the HALO plugin:

```
halo {
    services {
        translations true
    }
}
```

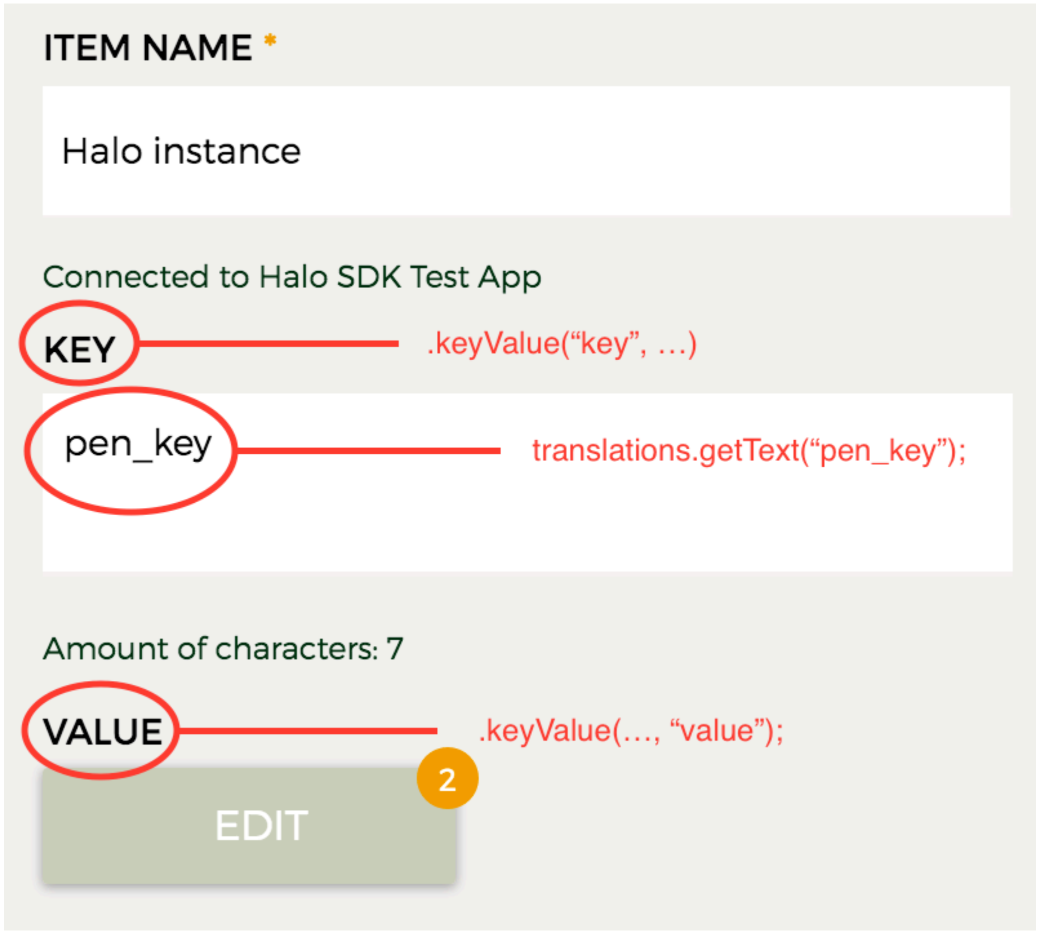
How can I use it?

Create a the plugin instance

It is fairly recommendable to create the instance as a singleton in your application class or using Dagger after installing HALO.

To make this work with HALO you will need to configure the module in the following way:

- A module created in HALO and its *module name*.
- Create as a manager two fields for this module, *one as a text* and a *second as a localized text*.
- Create some instances for this module with some localized texts.



To create the Android Application translations instance you will have to do the following:

```
Halo halo = Halo.installer(context).install();
HaloTranslationsApi translations = HaloTranslationsApi.with(halo)
                .locale(HaloLocale.ENGLISH_UNITED_STATES)
                .keyValue("key", "value")
                .moduleName(moduleName)
                .defaultText("No translation found")
                .defaultLoadingText("Translating...")
                .provideDefaultOnAsync(true)
                .build();
translations.load();
```

As you can see there are some parameters that you can provide to configure your translations plugin. In the following list you will find an explanation for each of those:

- **Locale** (Mandatory): This field specifies which is the locale that will be cached. You can change it later in the same instance using the `changeLocale()` method.
- **KeyValue** (Mandatory): To let the plugin know which of the values for the instance should be cached, you have to provide the name of the field that serves as a key and the name of the value that serves as a localized text. Afterwards you will be able to load a text by the key name in a TextView view.
- **ModuleName** (Mandatory): The name of the module. This name will be used to select which module should be synchronized. You can have as many modules as you want for localized texts as long as each module has its own instance of `HaloTranslationsApi` .
- **DefaultText** (Optional): You can provide either a default text or a strategy callback to select which text will be used as default in the case this text is not present in the module provided for any reason (no connection, someone removed it…). This parameter is optional.
- **DefaultLoadingText** (Optional): You can also optionally select a text that will be displayed when the translations are being loaded.
- **ProvideDefaultOnAsync** (Optional): The callback that provides the texts allows also to get the default value when loading. While loading the values it will call the text ready listener with the default value meanwhile the real one is being loaded.

Once the instance is created you can load the text as Strings or using a direct TextView. You don’t have to worry about memory leaks, rotations or so, just use your key, the plugin handles that for you. In the following snippet you can have a look how to do it:

```
translationsInstance.textOn(textView, keyName);
```

or

```
mTranslationApi.getTextAsync(keyName, new TextReadyListener() {
    @Override
    public void onTextReady(@Nullable String key, @Nullable String t
ext) {

    }
});
```

You can also use one of the following methods to interact with the translations api.

Method name	Explanation
`cancel`	Cancels the query done to sync the process.
`changeLocale`	Changes the locale in the current instance with another removing the previous translations and loading the new ones again.
`clearCallbacks`	Clears the pending callbacks.
`clearTranslations`	Clears the in memory translations. The ones in the database will remain.
`clearCachedTranslations`	Clears the in database and in memory translations.
`clearAll`	Clears callbacks and in mememory translations.
`getAllTranslations`	Provides a list of all the texts in the in memory map.
`getInMemoryTranslations`	Provides a map copy for the items in memory.
`getDefaultText`	Get the default text for the given key.
`getText`	Gets the in memory text for the given key.
`getTextAsync`	Gets the text as an async process to make sure it is loaded when it is provided.
`textOn`	Assigns a text on a text view when it is ready. It internally used the `setTextAsync` .
`load`	Loads asynchronously and sync the pending texts.
`isLoading`	Tells if the plugin is still syncing the data.
`locale`	Provides the current locale configured.
`moduleName`	Provides the module name for this translations instance.

Method name	Explanation
`setErrorListener`	Listen for possibly errors.
`removeLoadCallback`	Removes a loading callback in case you are in a context dependent environment.

Android SDK - Auth SDK Overview

Add dependency

In the HALO plugin add the following to enable the social sdk.

```
apply plugin: 'halo'

halo {
        ...
        services {
            auth {
                google "GOOGLE_CLIENT_ID"
                facebook "FACEBOOK_APP_ID"
            }
        }
        ...
}
```

Note: The provider credentials are optional so you may include only the social provider you want to use.

Social API

The social API is the way to sign in with social providers on HALO SDK. Importing this library will need a valid HALO instance configured with some credentials and the credentials of the network providers you want to import. The HALO Social SDK allows the user to sign in in three ways:

- HALO username and password.
- Facebook integration. If you want go in deep, please refer to the detailed documentation (page 87)
- Google plus integration. If you want go in deep, please refer to the detailed documentation (page 90)

It is fairly recommendable to create the instance as a singleton in your application class or using Dagger after installing HALO. Creating an instance of the Social API is really simple once you have your HALO running. Just write the following lines:

```
HaloAuthApi authApi = HaloAuthApi.with(haloInstance)
                .recoveryPolicy(HaloAuthApi.RECOVERY_ALWAYS)
                .storeCredentials("halo.account.demoapp")
                .withHalo()
                .withFacebook()
                .withGoogle()
                .build();
```

Also you have to release the memory in the onTerminate method of your application:

```
authApi.release();
```

As you can see there are some parameters that you can provide to configure your social instance. In the following list you will find an explanation for each of those:

Parameter name	Explanation
recoveryPolicy (Optional)	This field specifies if HALO will store your credentials with Android account manager. By default is set to HaloAuthApi.RECOVERY_NEVER so HALO will not store credentials.
storeCredentials (Optional)	This field specifies which is the account type to store on Android account manager. It is mandatory if recoveryPolicy is set to HaloAuthApi.RECOVERY_ALWAYS.
withHalo (Optional)	To use HALO as provider to login or sign in.
withFacebook (Optional)	To use facebook as provider to login.
withGoogle (Optional)	To use google as a provider to login.

Simple use

Login with HALO

Once the instance is created you can login with username and password only if the `wihtHalo()` was especified on the `HaloAuthApi` instance. This will try to login the user with this credentials(username,password):

```
//set a authentication profile to login
HaloAuthProfile authProfile = new HaloAuthProfile(username,passwor
d);
//set a callback
CallbackV2<IdentifiedUser> callback = new CallbackV2<IdentifiedUse
r>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<IdentifiedUser> r
esult) {
            //handle response
        }
    };
//request login with the authoritation profile
authApi.loginWithHalo(HaloAuthApi.SOCIAL_HALO, authProfile, callbac
k);
```

Note: The third parameter is the callback of type `CallbackV2<HaloUserProfile>` in wich you will handle the result of the authentication query.

Login with a social provider (Facebook or Google)

Once the instance is created you can login with social network access token only if the `wihtFacebook()` or `withGoogle()` was especified on the `HaloAuthApi` instance. This will try to login the user after the system obtain the social provider accessToken with the providers:

Warning: If you want go in deep on Facebook integration, please refer to the detailed documentation (page 87)

Warning: If you want go in deep on Google integration, please refer to the detailed documentation (page 90)

If you set `withFacebook()` :

```
//set a callback
CallbackV2<IdentifiedUser> callback = new CallbackV2<IdentifiedUse
r>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<IdentifiedUser> r
esult) {
            //handle response
        }
    };
authApi.loginWithSocial(HaloAuthApi.SOCIAL_FACEBOOK, callback);
```

If you set `withGoogle()` :

```
//set a callback
CallbackV2<IdentifiedUser> callback = new CallbackV2<IdentifiedUse
r>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<IdentifiedUser> r
esult) {
            //handle response
        }
    };
authApi.loginWithSocial(HaloAuthApi.SOCIAL_GOOGLE_PLUS, callback);
```

Note: The second parameter is the callback of type `CallbackV2<HaloUserProfile>` in wich you will handle the result of the authentication query.

Register

Once the instance is created you can register on HALO providing authoritation object and a user profile only if the `withHalo()` was especified on the `HaloAuthApi` instance. This will try to register the user with HALO:

Tip: This process only register the user against HALO so you must call to login after registration process finishes correctly.

```
//the authentication profile for the user
HaloAuthProfile authProfile = new HaloAuthProfile(username,passwor
d);
//the user profile to register
HaloUserProfile userProfile = new HaloUserProfile(null, displayNam
e, username, password, photoUrl, email);
//make registration with auth profile and user profile given.
authApi.register(authProfile,userProfile)
    .execute(new CallbackV2<HaloUserProfile>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<HaloUserProfile>
result) {
            if (result.status().isOk()) {
                // Handle result
            }
        }
    });
```

Note: You will handle the result of the registration process with a `CallbackV2<HaloUserProfile>` as a parameter of execute.

Check a provider

Once the instance is created you can check if a given provider is available. A provider will be available if it was defined when creating the instance and the library of the provider is available.

```
//the authentication profile for the user
authApi.isAuthProviderAvailable(HaloAuthApi.SOCIAL_HALO);
```

Release

You could release the memory in the `onTerminate` method of your application.

```
//release resources
authApi.release();
```

Halo Pocket API: Identified user data

With this new `HaloPocketAPI` you can store two new objects on the identified users collection. One for references `ReferenceContainer` and one for custom data that can be any model of your business or by default as `JSONObject` .

- To fetch all `Pocket` information you should use the `get()` method.

```
//get the pocket api instance
HaloPocketApi pocketApi = authApi.pocket();
//get the pocket data (user custom data and filter references)
pocketApi.get().execute(new CallbackV2<Pocket>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<Pocket> resul
t) {

            }
        });
```

- To store all `Pocket` information you should use the `save(Pocket pocket)` method.

```
//get the pocket api instance
HaloPocketApi pocketApi = authApi.pocket();
//create a pocket instance
Pocket pocket = new Pocket.Builder()
                .withData(customModelClass)
                .withReferences(referenceContainer)
                .build();
//get the pocket data (user custom data and filter references)
pocketApi.save(pocket).execute(new CallbackV2<Pocket>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<Pocket> resul
t) {

            }
        });
```

Warning: If you want go in deep, please refer to the detailed documentation (page 93)

Android SDK - Integration with Facebook

This getting started guide will guide you on setting up Google SDK for Android in a few minutes. We will provide a step by step guide to get everything working with the most basic setup.

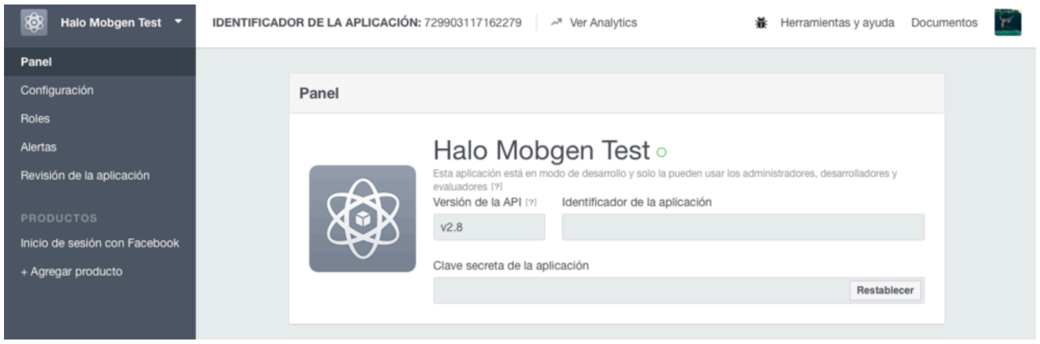
> **Warning:** You don't need to import Facebook Android SDK. It's automatically done by the HALO plugin.

Step 1: Create the app

Register in the facebook console and create a new app. You must have a properly configured developer account.

Optional Step to add security

> **Note:** You can add extra security if you add the client secret and client id into the Halo CMS.



Esta aplicación está en modo de desarrollo y solo la pueden usar los administradores, desarrolladores y evaluadores [?]

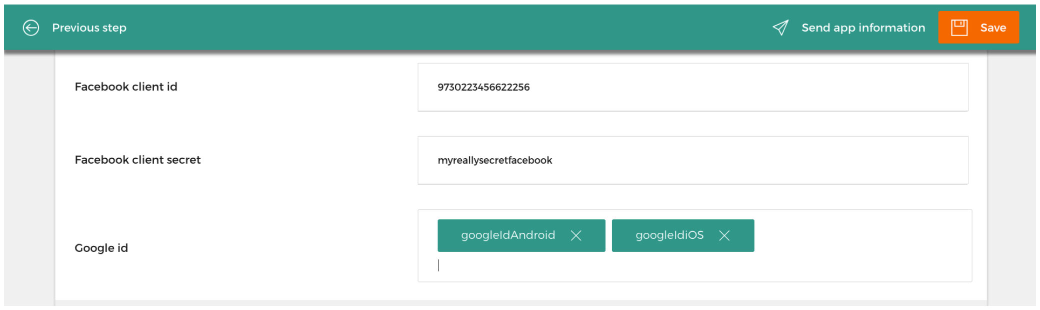
Versión de la API [?]: v2.8

Identificador de la aplicación

Clave secreta de la aplicación

This step is optional and if you add this field to the HALO CMS it would verify that the tokens you provide belongs to the Facebook application. In another case the HALO system only verifies if it is a valid token against Facebook.

You can add this information in app section on HALO CMS.



Facebook client id	9730223456622256
Facebook client secret	myreallysecretfacebook
Google id	googleIdAndroid, googleIdiOS

Step 2: Add your package

Add the package name and your potential deeplink activity on the facebook console.

Step 3: Generate the hashes

To generate a hash of your release key, run the following command substituting your release key alias and the path to your keystore.

```
keytool -exportcert -alias <RELEASE_KEY_ALIAS> -keystore <RELEASE_KE
Y_PATH> | openssl sha1 -binary | openssl base64
```

This command should generate a string. Copy and paste this Release Key Hash into your facebook console.

Step 4: Configure HALO

To enable facebook integration on HALO you must use the FACEBOOK_APP_ID:

```
apply plugin: 'halo'

halo {
    ...
    services {
            auth {
                facebook "FACEBOOK_APP_ID"
            }
    }
    ...
}
```

Step 5: Enable single sign-on

Open the app in the console, open settings and enable “Single sign-on” by setting it to YES. Make sure you save the changes.

Step 6: Create the halo auth instance

Create the `HaloAuthApi` instance login with facebook. It is fairly recommendable to create the instance as a singleton in your application class.

```
HaloAuthApi authApi = HaloAuthApi.with(halo)
                .withFacebook()
                .build();
```

Step 7: Login with Facebook

With the `HaloAuthApi` instance login with facebook provider.

```
CallbackV2<IdentifiedUser> callback = new CallbackV2<IdentifiedUse
r>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<IdentifiedUser> r
esult) {
            //handle response
        }
    };
authApi.loginWithSocial(HaloAuthApi.SOCIAL_FACEBOOK, callback);
```

Note: For further information about Facebook SDK visit the official Facebook documentation page (https://developers.facebook.com/docs/facebook-login/android)

Android SDK - Integration with Google

This getting started guide will guide you on setting up Google Sign-in SDK for Android in a few minutes. We will provide a step by step guide to get everything working with the most basic setup.

Warning: You don't need to import Firebase Android SDK. It's automatically done by the HALO plugin.

Step 1: Create the app

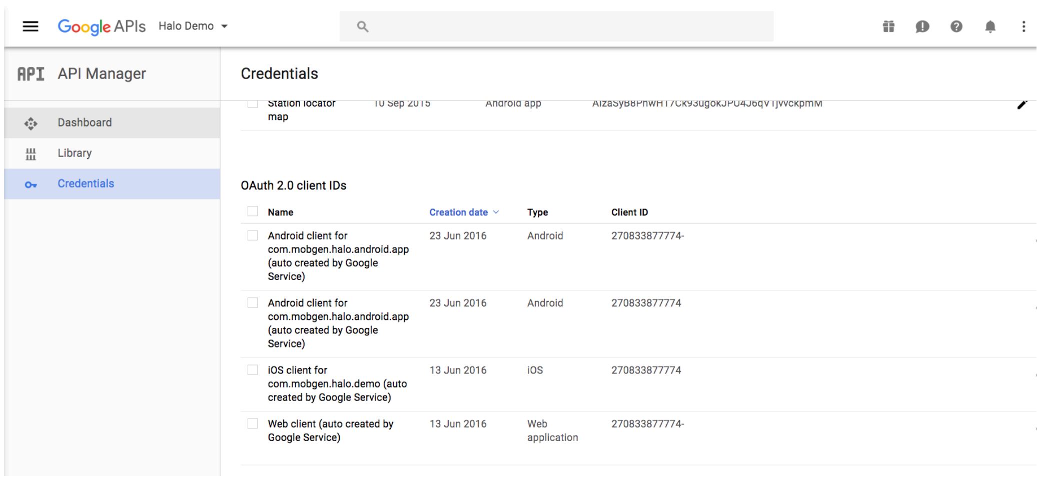
Register in the firebase console and create a new app, if you don't already have one. If you already have an existing Google project associated with your mobile app, click Import Google Project. Otherwise, click Create New Project.

Step 2: Generate an oAuth web key

Add the package name and and follow the setup steps.

Optional Step to add security

Note: You can add extra security if you add the google id into the Halo CMS.



Credentials

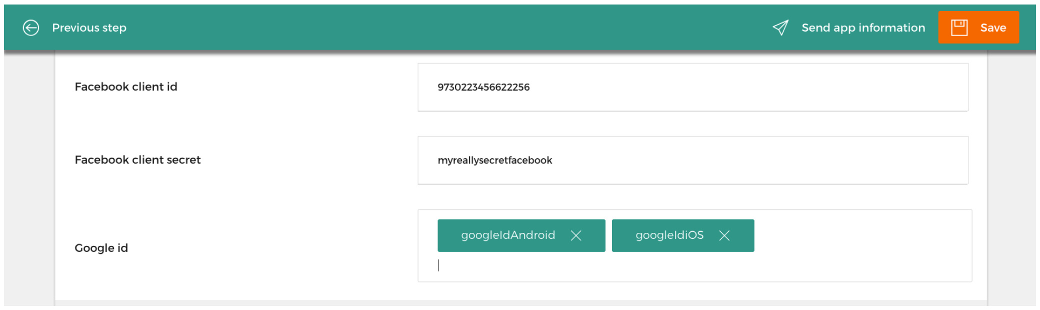
Station locator map	10 Sep 2015	Android app	AIzaSyB8PnWHT7Ck93ugoKJPU4J6qVTjWckpmM

OAuth 2.0 client IDs

Name	Creation date	Type	Client ID
Android client for com.mobgen.halo.android.app (auto created by Google Service)	23 Jun 2016	Android	270833877774-
Android client for com.mobgen.halo.android.app (auto created by Google Service)	23 Jun 2016	Android	270833877774
iOS client for com.mobgen.halo.demo (auto created by Google Service)	13 Jun 2016	iOS	270833877774
Web client (auto created by Google Service)	13 Jun 2016	Web application	270833877774-

This step is optional and if you add this field to the HALO CMS it would verify that the tokens you provide belongs to the Google application. You can provide as many ids as you need for different platforms. In another case the HALO system only verifies if it is a valid token against Google.

You can add this information in app section on HALO CMS.



Facebook client id	9730223456622256
Facebook client secret	myreallysecretfacebook
Google id	googleIdAndroid, googleIdiOS

Step 3: Generate the hashes

To generate a hash of your release key, run the following command substituting your release key alias and the path to your keystore.

```
keytool -exportcert -alias <RELEASE_KEY_ALIAS> -keystore <RELEASE_KE
Y_PATH> | openssl sha1 -binary | openssl base64
```

This command should generate a string. Copy and paste this Release Key Hash into your firebase console.

Step 4: Download and setup configuration file

At the end, you'll download a google-services.json file. You can download this file again at any time. Copy google-services.json into your apps main folder.

Step 5: Configure HALO

To enable google signin integration on HALO you must use the GOOGLE_CLIENT_ID:

```
apply plugin: 'halo'

halo {
    ...
    services {
            auth {
                google "GOOGLE_CLIENT_ID"
            }
    }
    ...
}
```

Step 6: Create the halo auth instance

Create the `HaloAuthApi` instance to login with google. It is fairly recommendable to create the instance as a singleton in your application class.

```
HaloAuthApi authApi = HaloAuthApi.with(halo)
                .withGoogle()
                .build();
```

Step 7: Login with Google

With the `HaloAuthApi` instance login with google provider.

```
CallbackV2<IdentifiedUser> callback = new CallbackV2<IdentifiedUse
r>() {
        @Override
        public void onFinish(@NonNull HaloResultV2<IdentifiedUser> r
esult) {
            //handle response
        }
    };
authApi.loginWithSocial(HaloAuthApi.SOCIAL_GOOGLE_PLUS, callback);
```

Note: For further information about Firebase SDK visit the official Firebase documentation page (https://firebase.google.com/docs/android/setup)

Android SDK - Pocket SDK Overview

Pocket API

The pocket API is the way to store and fetch data from identified users. Importing this library will need a valid HALO instance configured and the you will need a valid identified user credentials to create pocket request.

It is fairly recommendable to create the instance as a singleton in your application class or using Dagger after installing HALO. Creating an instance of the Pocket API is really simple once you have your HALO running. Just write the following lines:

```
HaloPocketApi pocketApi = HaloPocketApi.with(haloInstance)
                .build();
```

Also you can get a instance of HaloPocketApi from the Auth API as follows:

```
HaloPocketApi pocketApi = authApi.pocket();
```

As you can see there are some methods that you can use to get or save the pocket data. In the following list you will find an explanation for each of those:

Method name	Explanation
get	It does the request with ReferenceFilter.all() and fetches all custom data and filter references.
getData	It does the request with ReferenceFilter.noReferences() to avoid fetching all the references.
getReference	Fetch all filter references but without all user custom data.
save	Update the pocket data and references.
saveData	Update the custom data of the pocket.
saveReference	Update the references of the pocket.

Simple use

You can fetch the data in several ways (all, only references, only data as JSON or only data as your custom model).

First get the `HaloPocketApi` instance

```
HaloPocketApi pocketApi = authApi.pocket();
```

Get data from identified user.

Get all data

```
//get the pocket data (user custom data and filter references)
pocketApi.get().execute(new CallbackV2<Pocket>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<Pocket> resul
t) {
               //handle the result
            }
        });
```

Get references

```
//set the filter references
ReferenceFilter referenceFilter = new ReferenceFilter.Builder().filt
ers("favorites").build();
//get the pocket data (user custom data and filter references)
pocketApi.getReferences(referenceFilter).execute(new CallbackV2<Lis
t<ReferenceContainer>>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<List<Referenc
eContainer>> result) {
                //handle the result
            }
        });
```

Get data as custom model

```
pocketApi.getData()
        .asCustomData(MyModel.class)
        .execute(new CallbackV2<MyModel>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<MyModel> resu
lt) {
                //handle the result
            }
        });
```

Get data as Pocket

```
pocketApi.getData()
        .asPocket()
        .execute(new CallbackV2<Pocket>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<Pocket> resul
t) {
                //handle the result
            }
        });
```

Update identified user data.

Save all data

```
//create the pocket
Pocket pocket = new Pocket.Builder()
                .withData(userDummy)
                .withReferences(referenceContainer)
                .build();
//get the pocket data (user custom data and filter references)
pocketApi.save(pocket).execute(new CallbackV2<Pocket>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<Pocket> resul
t) {
               //handle the result
            }
        });
```

Save references

To save references you will have the following options:

- If you set a array with content you will modify this reference or create a new one if this one doesn't exists.
- If you set an empty array into the list of references on the ReferenceContainer then you will empty this reference.
- If you set to null the list of references on the ReferenceContainer then you will delete this reference.

```
//set the filter references
ReferenceContainer referenceContainer = new ReferenceContainer("myco
llection", null);
//get the pocket data (user custom data and filter references)
pocketApi.saveReferences(referenceContainer).execute(new CallbackV
2<List<Pocket>>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<List<Pocke
t>> result) {
                //handle the result
            }
        });
```

Save data as custom model

```
pocketApi.saveData(myModelClassInstance)
        .execute(new CallbackV2<Pocket>() {
            @Override
            public void onFinish(@NonNull HaloResultV2<Pocket> resul
t) {
                //handle the result
            }
        });
```

Android SDK - Two Factor Authentication SDK Overview

Add dependency

In the HALO plugin add the following to enable the two factor authentication sdk.

```
apply plugin: 'halo'

halo {
        ...
        twofactorauth {
            push true
            sms true
        }
        ...
}
```

Note: You can enable sms or push notification two factor confirmation.

Two Factor Authentication API

The Two Factor Authentication API is the helper to confirm you authentication code on HALO SDK. Importing this library will need a valid HALO instance configured (it will import push notification automatically if needed). The HALO Two Factor Authentication SDK allows the user to listen to confirmation codes in a simple listener from two providers:

- Push notification. If you want go in deep about notifications, please refer to the detailed documentation (page 60)
- SMS notification.

It is fairly recommendable to create the instance as a singleton in your application class or using Dagger after installing HALO. Creating an instance of the Two Factor Authentication API is really simple once you have your HALO running. Just write the following lines:

```
HaloTwoFactorApi twoFactorAuthentication = HaloTwoFactorApi.with(hal
o)
                .smsProvider("HALO")
                .withNotifications(mHaloNotificationApi)
                .withSMS()
                .build();
```

Also you have to release resources when you are done:

```
twoFactorAuthentication.release();
```

As you can see there are some parameters that you can provide to configure your two factor authentication instance. In the following list you will find an explanation for each of those:

Parameter name	Explanation
smsProvider (Optional)	Set the name of the sms sender. By default it is set to HALO.
withNotifications (Optional)	Prepare the two factor auth listener to receive codes from push notifications. You must provide a valid HALO notification API.
withSMS (Optional)	Prepare the two factor auth listener to receive codes from SMS notification.

Simple use

Listen two factor attempts

Once the instance is created you can listen to Two Factor Api to receive a `TwoFactorCode` object. This object store the two factor code and the issuer (SMS or push) of the code. You will handle two types of issuers:

- `HaloTwoFactorApi.TWO_FACTOR_NOTIFICATION_ISSUER`
- `HaloTwoFactorApi.TWO_FACTOR_SMS_ISSUER`

```
twoFactorAuthentication.listenTwoFactorAttempt(new HaloTwoFactorAtte
mptListener() {
            @Override
            public void onTwoFactorReceived(@NonNull TwoFactorCode t
woFactorCode) {
                if(twoFactorCode.getIssuer().equals(HaloTwoFactorAp
i.TWO_FACTOR_NOTIFICATION_ISSUER)){
                    //handle when comes from a push notification
                    //you could show a notification on the bar.
                    showNotificationOnTheBar();
                } else {
                    Toast.makeText(context, "This is the code receiv
ed: " +
                    twoFactorCode.getCode() + " from a: " + twoFacto
rCode.getIssuer(), Toast.LENGTH_LONG).show();
                }

            }
        });
```

Note: If you are using the push notification provider remember that push notifications are silents so it is your responsability to show them if you want.

Note: If you are using the SMS provider remember to set the SMS provider name to listen to the two factor authentication codes.

Release

You should release the memory when you are done. After release the attempt listener is not available anymore.

```
//release resources
twoFactorAuthentication.release();
```

Android SDK - Presenter SDK Overview

Download 2.5.2 (https://bintray.com/halo-mobgen/maven/HALO-Presenter/_latestVersion)

This library is intended to help an application to implement and attach the presenter lifecycle with the Activity or Fragment one. It provides a default implementation of the presenter called `EmptyPresenter` if you want to have an activity without a real presenter.

The MVP architecture is widely used when creating interfaces and having a helper in Android is really useful. Follow the next sections to learn how to use it and which are the functionalities provided.

In MVP there are three roles:

- The model: represents the content that will be displayed.
- The view: contains the UI elements and all the logic related to views and they way they are displayed.
- The presenter: brings the data and calls the needed methods from the view to allow the bridge between real business logic and the presentation logic.

Setup the lib

You can use the HALO plugin to configure it in an easy way:

```
halo {
        ...
        services {
                presenter true
        }
        ...
}
```

Since it is really well integrated with HALO, it provides in the presenter a method called `onInitialized` called once halo is ready. This can help us to show something while HALO is being prepared and also having a notification to perform some actions with it.

We provide also an interface called `HaloViewTranslator` that has some typical methods like `startLoading` , `stopLoading` or `showError` . This view translator should be implemented by your activity or other interfaces that inherits from it to provide functionality to the presenter.

To persist data also in the bundle the presenter provides two methods: `onSaveInstanceState` and `onInitStarted`, both managing the bundles on which the data is stored and retrieved respectively.

You can also inherit from the `AbstractHaloPresenter` to avoid reimplementing everyting and because it has already implemented the support for network conection drops and presenter lifecycle management.

How can I use it?

1. Inherit from one of `HaloActivity`, `HaloActivityV4`, `HaloFragment` or `HaloFragmentV4`.
2. Create a presenter that inherits from `AbstractHaloPresenter`.
3. Create an interface for your view that inherits from `HaloViewTranslator`.
4. Make your activity implement that interface.
5. Start coding your MVP logic in both, presenter and view.