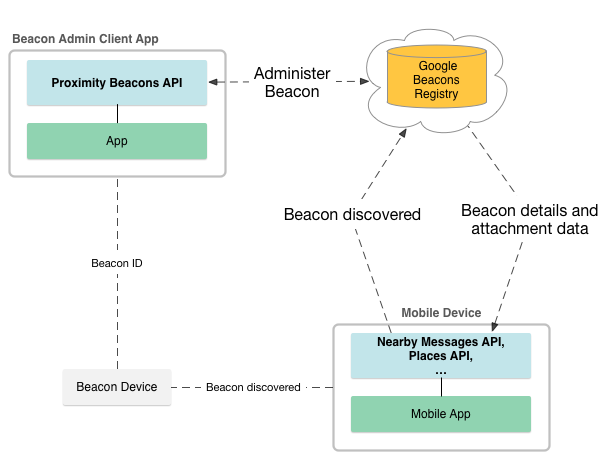
# Eddystone Beacon Android Documentation



**High Level Steps**

1. Provision beacon using your beacon manufacturer’s method
2. Set up beacon using Proximity API and a REST api client
3. Create mobile client app leveraging Nearby Messages, plcase, or other API to subscribe and interact with the beacon

**Proximity API –** Proximity API is a web service used to administer data associated with a beacon, such as meta data and attachments. Proximity API uses a REST interface, google offers OAuth Playground that you can use as a REST client (<https://developers.google.com/oauthplayground/>).

**Nearby Messages API –** Nearby Messages API is used by beacons to publish their attachments stored on Google Beacons Registry and given to them through the Proximity API method. A mobile device can use Nearby Messages API to subscribe, enabling the device to receive the messages from beacons within range.

The following sequence shows the events leading to message exchange between Beacon and App:

1. A publishing app in the form of a beacon makes a request to associate a binary payload (the message) with a unique-in-time pairing code (token). The server makes a temporary association between the message payload and the token, this is covered in the next steps below of attaching data to a beacon.
2. The beacon uses a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and an ultrasonic modem to make the token detectable by nearby devices.
3. A subscribing app associates its subscription with a token and uses a mix of the above technologies to send its token to the publisher, and to detect the publisher's token.
4. When either side detects the other's token, it reports it to the server.
5. The server facilitates message exchange between two devices when both are associated with a common token, and the API keys used by the calling apps are associated with the same project in the [Google Developers Console](https://console.developers.google.com/).

**Guide Using Proximity API**

1. Provision Beacon

* + Provision beacon includes setting namespace and instance ID, Google Beacons Registry uses this to send the correct attachment to a device that has subscribed using Nearby Messages API within range if the beacon
  + In order to do this step the manufacturer of your beacon will provide a method to set these values if they do not come preset. For example for BEEKS beacons there is a mobile app.

2. Prepare google developer project

* NOTE: Beacons can only be registered with ONE google project at a time, if a beacon needs to be changed to a different project then they must be decommissioned using Proximity API, re-provisioned with a new ID, and then registered to the new project with the new ID. However, the old ID can never be used again.
  + In order to register and administer your beacon you must use the Proximity rest API and in order to use that a google project must be created and the proximity API added to it as well as a client ID
  + These steps will use the google OAuth Playground (https://developers.google.com/oauthplayground/)
  + First create a project in the developer console: <https://console.developers.google.com/home/>, or if you have one that is already created that you would like to use you can skip this step.
  + Request an OAuth 2.0 Client ID
  + Requests which can modify beacon data, or return administrative beacon data, must be authorized using OAuth 2.0. Follow these steps to request an OAuth 2.0 client ID for the Google Proximity Beacon API.
  + Go to the Google Developers Console: https://console.developers.google.com/flows/enableapi?apiid=proximitybeacon
  + Select a project, or create a new one that you would like to use.
  + Click Continue to enable the Google Proximity Beacon API.
  + Click Go to credentials.
  + Select blue link that says "client ID".
  + Under Application type, select the appropriate choice for your app (Web application, Android, iOS, etc.). Select Web application to get credentials you can use with the OAuth 2.0 Playground.
  + You will be prompted for further information based on the Application type you chose. Fill in the appropriate fields.
  + Click Create to get your new OAuth 2.0 Client ID. The ID appears as a string of characters that looks something like this:

780816631155-gbvyo1o7r2pn95qc4ei9d61io4uh48hl.apps.googleusercontent.com

* Request a browser API key
* Requests for public data (for example calls to methods like beaconinfo.getforobserved) do not require authorization, but must still be authenticated by using an API key. Follow these steps to request a browser API key for the Google Proximity Beacon API.
* Go to the Google Developers Console.
* Select the project used in the last steps.
* Go to credentials, should be in the menu to the left of the screen.
* Click create credentials, then select API Key.
* Under Create a new key select Browser key.
* Enter a name for the key.
* Under Accept request from these HTTP referrers (web sites), enter the base URL of the site from which requests will originate. You can skip this step if the key will only be used for testing.
* Click Create. Your new API key will appear in the list of API keys for your project.

3. Try the REST API in the OAuth Playground

* The OAuth Playground is the easiest way to familiarize yourself with the Google Proximity Beacon API by submitting HTTP requests and observing the responses.
* If you have not done so already, follow the instructions above to create a new Web application client ID.
* Go to the OAuth 2.0 Playground: (https://developers.google.com/oauthplayground/)
* Click the gear icon to open the configuration options and make the following changes:
  + Set OAuth flow to Client-side.
  + Check Use your own OAuth credentials.
  + Paste in your OAuth client ID.
  + Close the configuration options.
* Under Step 1 Select & authorize APIs, paste the scope for the Proximity Beacon API into the field labeled Input your own scopes: https://www.googleapis.com/auth/userlocation.beacon.registry
* Click Authorize APIs.
* Click Allow when prompted.
* Under Step 2 Configure request to API, paste the following URI into the Request URI field: https://proximitybeacon.googleapis.com/v1beta1/beacons
* Set HTTP Method to GET.
* Click Send the request.
* Nothing should show up because we have not registered any beacons, so you should see a resource not found error.

4. Registering beacon

* Using playground we will register a beacon with your google project.
* NOTE: A beacon ID can only be associated with one project, the only way to use a beacon with another project is to decommission the beacon through Proximity API, provision a new ID, and then re-register the beacon using the new ID. However, the old ID can never be used again.
* Use the URI: https://proximitybeacon.googleapis.com/v1beta1/beacons:register
* Set it to a post method
* For the request body you must use the format given in this documentation: https://developers.google.com/beacons/proximity/reference/rest/v1beta1/beacons/register
  + NOTE: The "id" in the "advertisedId" must be the base64 encoded string from the concatination of the namespaceID and instanceID associated with your beacon, so encode it using a 3rd party program.
* an example is given below:

{

"advertisedId": {

"type": "EDDYSTONE",

"id": "Fr4Z98nSoW0hgAAAAAAAAg=="

},

"status": "ACTIVE",

"placeId": "ChIJTxax6NoSkFQRWPvFXI1LypQ",

"latLng": {

"latitude": "47.6693771",

"longitude": "-122.1966037"

},

"indoorLevel": {

"name": "1"

},

"expectedStability": "STABLE",

"description": "An example beacon.",

"properties": {

"position": "entryway"

}

}

* The response should send the Beacon information with a 200 code response
* Now if you call the rest API https://proximitybeacon.googleapis.com/v1beta1/beacons in a GET method you should now see your beacon

5. Attaching data to the beacon

* You can associate arbitrary data (Up to 1024 bytes long) with a beacon using an attachment. Attachments are stored as blobs in Google’s scalable cloud. You can retrieve existing attachments via an API request, for example as messages using the Nearby Messages API.
* Once a beacon has been registered it is ready to accept attachments. The Proximity Beacon API makes it easy to add and manage attachments on registered beacons.
* When you create an attachment, you must populate two fields:
  + namespacedType: a string made up of a namespace identifier, followed by a forward slash and the data type. For example, surreptitious-banjo-145/string.
  + data: a base64 encoded value of the data type defined in the namespacedType field. For example, aGVsbG8gd29ybGQh.
* To find out which namespaces are associated with your project, call namespaces.list (<https://developers.google.com/beacons/proximity/reference/rest/v1beta1/namespaces/list>) using Proximity API.
* Attachments can be up to 1024 bytes long. You can use any string that has meaning to your app, for example the ID for a bus stop or store location, structured data such as JSON, or a reference to an external datastore.
* Encode the data to be attached as base64 the same way you encoded the beacon advertisedID
* This example demonstrates how to attach data (the base 64 encoded string "hello world!") to a beacon.
  + HTTP method: POST
  + Request URL: https://proximitybeacon.googleapis.com/v1beta1/beacons/beaconName/attachments
  + Request body:

{

"namespacedType":"your-name-space/my-attachment-type",

"data":"aGVsbG8gd29ybGQh"

}

**Creating Mobile Device Client App**

NOTE: The following code snippets are taken from a sample app provided by google, the link is as follows: https://github.com/googlesamples/android-nearby/tree/master/messages/NearbyDevices

1. Create android project or configure an existing one using this guide:

<https://developers.google.com/nearby/messages/android/get-started#step_4_configure_your_project>

* NOTE: activity must implement GoogleApiClient.ConnectionCallbacks,  
   GoogleApiClient.OnConnectionFailedListener

1. Retrieve Message From Beacon:

* First implement a message listener to process the message and any other action necessary for your functionality using the following format:

mMessageListener = new MessageListener() {

@Override

public void onFound(Message message) {

// Do something with the message here.

Log.i(TAG, "Message found: " + message);

Log.i(TAG, "Message string: " + new String(message.getContent()));

Log.i(TAG, "Message namespaced type: " + message.getNamespace() +

"/" + message.getType());

}

public void onLost(Message message) {

Log.i(TAG, "message lost: "+ new String(message.getContent()));

}

};

* + NOTE: The device does not receive the message multiple times, only when the device first finds the beacon and therefore the message, the message must be lost before it can be re-found. A message is lost when the device travels out of range of the beacon and therefore cases relating to multiple beacons being found at once must be handled accordingly.
* The Nearby Messages API requires user consent. Prior to making any requests, your app must check to see whether the user has consented, and present them with a consent dialog if they have not. To implement the runtime consent flow in your app, you can implement a method to handle prompting the user for consent, which is called if the subscribe method fails. An example of this method is given below:

private void handleUnsuccessfulNearbyResult(Status status) {

Log.i(TAG, "Processing error, status = " + status);

if (mResolvingError) {

// Already attempting to resolve an error.

return;

} else if (status.hasResolution()) {

try {

mResolvingError = true;

status.startResolutionForResult(getActivity(),

Constants.REQUEST\_RESOLVE\_ERROR);// REQUEST\_RESOLVE\_ERROR = 1001

} catch (IntentSender.SendIntentException e) {

mResolvingError = false;

Log.i(TAG, "Failed to resolve error status.", e);

}

} else {

if (status.getStatusCode() == CommonStatusCodes.NETWORK\_ERROR) {

Toast.makeText(getActivity().getApplicationContext(),

"No connectivity, cannot proceed. Fix in 'Settings' and try again.",

Toast.LENGTH\_LONG).show();

} else {

// To keep things simple, pop a toast for all other error messages.

Toast.makeText(getActivity().getApplicationContext(), "Unsuccessful: " +

status.getStatusMessage(), Toast.LENGTH\_LONG).show();

}

}

}

* Subscribing to Nearby Messages in the foreground
  + When your app subscribes to beacon messages in the foreground, scans are performed continuously until your app unsubscribes. Only start a foreground subscription when your app is active, typically in response to a user action.

// Subscribe to receive messages.

Log.i(TAG, "Trying to subscribe.");

// Connect the GoogleApiClient.

if (!mGoogleApiClient.isConnected()) {

if (!mGoogleApiClient.isConnecting()) {

mGoogleApiClient.connect();

}

} else {

SubscribeOptions options = new SubscribeOptions.Builder()

.setStrategy(Strategy.BLE\_ONLY)

.setCallback(new SubscribeCallback() {

@Override

public void onExpired() {

Log.i(TAG, "No longer subscribing.");

}

}).build();

Nearby.Messages.subscribe(mGoogleApiClient, mMessageListener, options)

.setResultCallback(new ResultCallback<Status>() {

@Override

public void onResult(Status status) {

if (status.isSuccess()) {

Log.i(TAG, "Subscribed successfully.");

} else {

Log.i(TAG, "Could not subscribe.");

// Check whether consent was given;

// if not, prompt the user for consent.

handleUnsuccessfulNearbyResult(status);

}

}

});

}

* + NOTE: in the onConnected method, which is called after the api client has connected, implement same code above, for example:

public void onConnected(Bundle bundle) {

Log.i(TAG, "APICLIENT Connected");

SubscribeOptions options = new SubscribeOptions.Builder()

.setStrategy(Strategy.BLE\_ONLY)

.setCallback(new SubscribeCallback() {

@Override

public void onExpired() {

Log.i(TAG, "No longer subscribing.");

}

}).build();

Nearby.Messages.subscribe(mGoogleApiClient, mMessageListener, options)

.setResultCallback(new ResultCallback<Status>() {

@Override

public void onResult(Status status) {

if (status.isSuccess()) {

Log.i(TAG, "Subscribed successfully.");

} else {

Log.i(TAG, "Could not subscribe.");

// Check whether consent was given;

// if not, prompt the user for consent.

handleUnsuccessfulNearbyResult(status);

}

}

});

}

* NOTE: In order to subscribe in the background, you must implement an IntentService with the onFound and onLost methods implemented in the onHandleIntent method of the IntenetService. Subscribe in the same way as the foreground but passing your implemented intent service instead of passing the message listener.
  + Example:

Nearby.Messages.subscribe(apiClient, getPendingIntent(), options)

.setResultCallback(new ResultCallback<Status>() {

@Override

public void onResult(Status status) {

if (status.isSuccess()) {

Log.i(TAG, "Background Subscribed successfully.");

} else {

Log.i(TAG, "Could not background subscribe.");

handleUnsuccessfulNearbyResult(status);

}

}

});

private PendingIntent getPendingIntent() {

PendingIntent output = PendingIntent.getService(getApplicationContext(), 0,

getBackgroundSubscribeServiceIntent(), PendingIntent.FLAG\_UPDATE\_CURRENT);

this.backgroundIntent = output;

return output;

}

private Intent getBackgroundSubscribeServiceIntent() {

return new Intent(getApplicationContext(), BackgroundSubscribeIntentService.class);

}

* To unsubscribe and stop receiving device messages, call Nearby.Messages.unsubscribe().

private void unsubscribe() {

Log.i(TAG, "Trying to unsubscribe.");

Nearby.Messages.unsubscribe(mGoogleApiClient, mMessageListener);

}

* + NOTE: if unsubscribing from the background then pass the instance of the background service class used to subscribe.