G
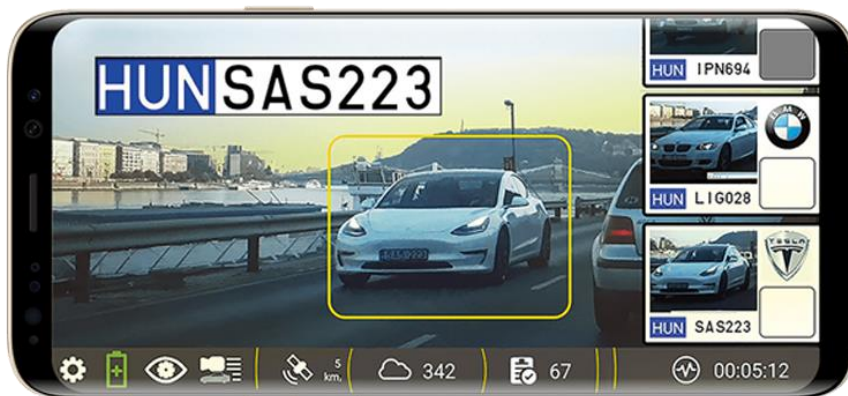
Carmen® Mobile
Integration Guide

This guide contains instructions on how to integrate Carmen® Mobile into any system.

RELEASE DATE

22.09.2023.

CARMEN MOBILE
INTEGRATION MANUAL

Document version: 2023.09.22.

Table of Contents

1. ABOUT CARMEN MOBILE & CARMEN CLOUD

1.1. BRIEF DESCRIPTION

Carmen Mobile is an app running on Android Devices. It is capable to record events from passing vehicles based on vehicle detection and Carmen Cloud ANPR. An event consists of an image and metadata like the full ANPR result, the event time and location.

1.2. SYSTEM ARCHITECTURE & BASIC LOGIC

From integration point of view, the basic logic is the following:

1. Based on an automatic or manual trigger, an image with a detected vehicle is sent to the Carmen Cloud.
2. The Cloud service reads the plate with the Carmen engine, and send the result to the Android device.
3. The device stores an event in its local database. (Image & metadata)
4. If uploading function is enabled, the device uploads the event to the Integrators system.
5. The integrator can store and process the event.
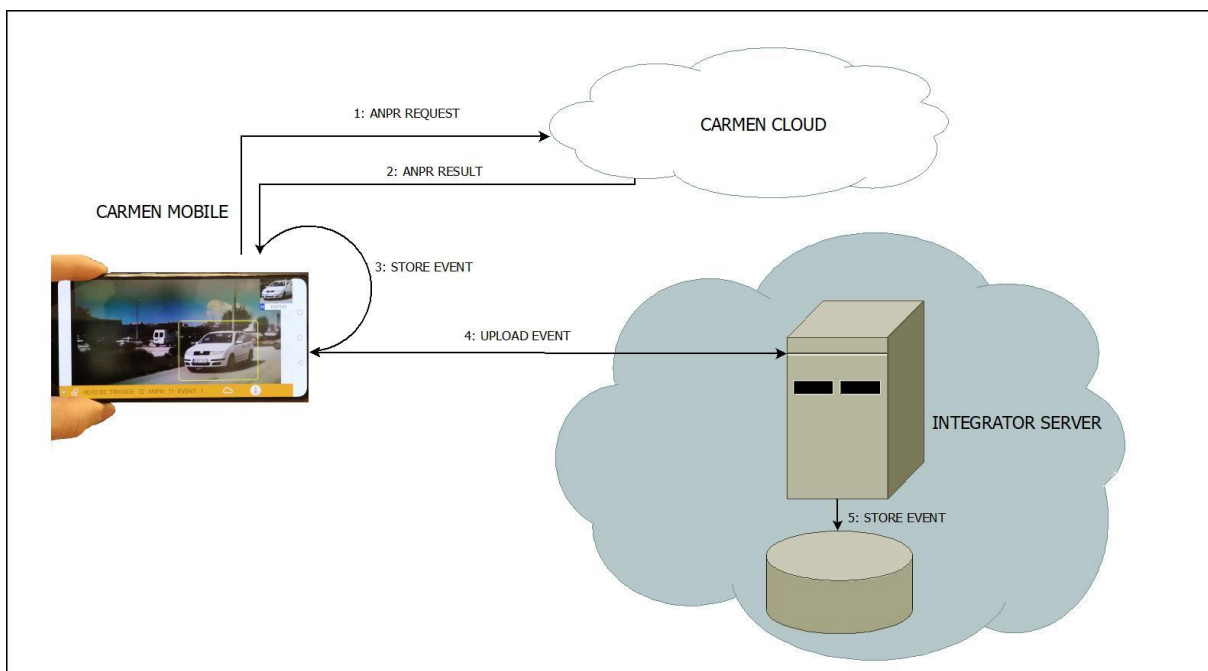
1.3. HOTLISTS OVERVIEW

There are three supported methods of hotlist handling.

1. You can manually add the plates in the applications' settings.
2. You can set up a server which can communicate with the app.
3. You can download a csv from a weblink. This can be generated by a back-office system.

After storing the event, the hotlist check is activated. The app checks its local database for a matching plate and it also uploads the plate to a configured Hotlist server. In a matching case, the app generates an alert. Two types of alert can be generated, deny and allow. When allow alert occurred, the corresponding plate color turns into green. In case of denylisted events, a warning sound is indicating, and a visual alert is created, with the image of the corresponding vehicle, and a blinking background.

This is the modified logic using a Hotlist server:
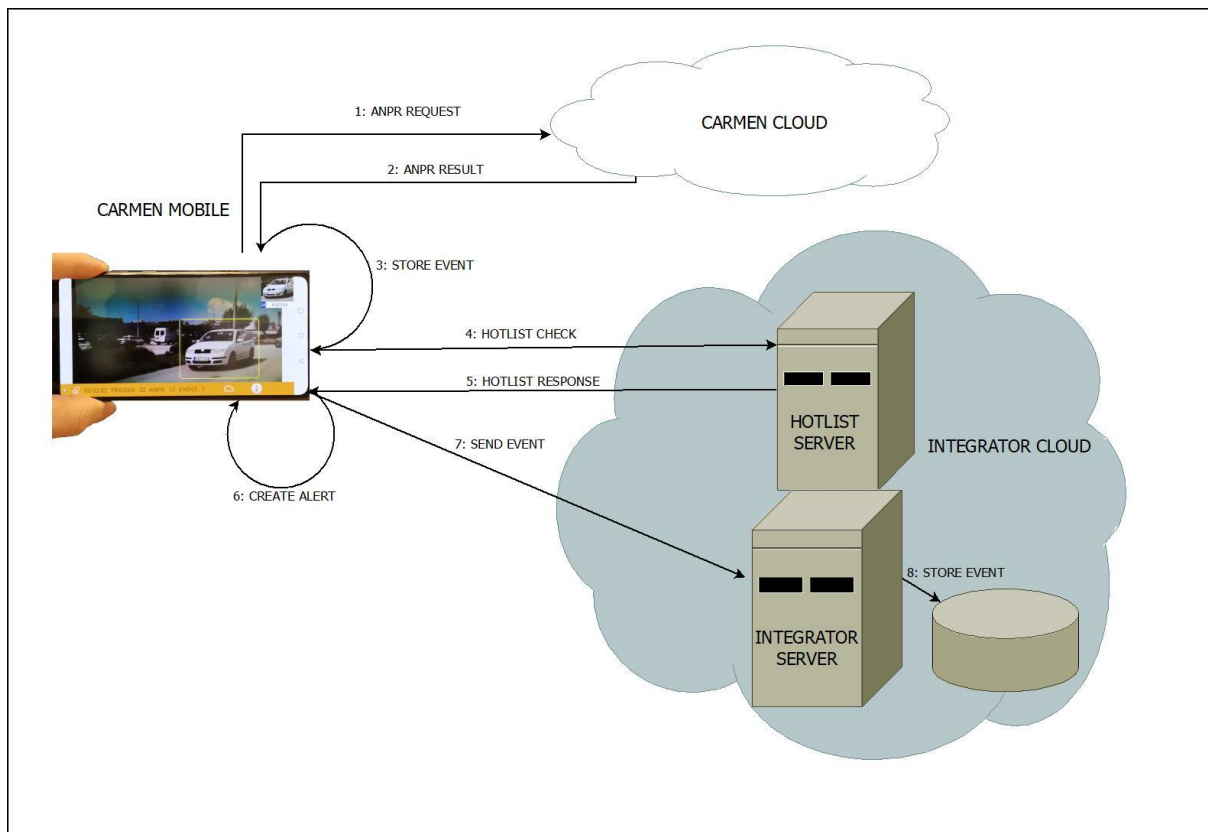
> 📱 Note
>
> Use Hotlist settings only if you need alert on your Carmen Mobile device, otherwise upload settings are recommended.

2. INTEGRATION OPTIONS

2.1. UPLOAD

There are multiple ways of uploading the events. In all cases you have to save a valid URL in the settings tab's upload path. You can scan the URL from a QR code as well. You have to enter the path of the server, and you can scan the generated QR code with the app. Go to /SETTINGS/Integration/Event upload/upload path and click ont he QR code for fast setup.

Here are some online QR-code generators.

https://www.qr-code-generator.com/

https://www.the-qrcode-generator.com/

You can select what kind of media you want to upload, in the upload settings.

2.1.1. On the fly upload

By selecting "Upload on the fly" option, the application will upload the events immediately after recording them. This is not a recommended feature unless you have a strong 4g connection.
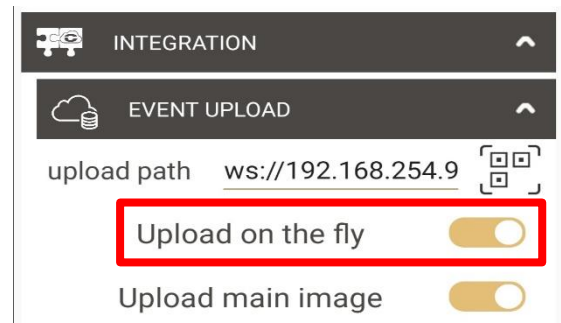
2.1.2. Uploading To GDS

The App can upload the events to GLOBESSEY® DATA SERVER – GDS. For this option you have to fill the "upload path" in the settings/integration/event upload, with a valid GDS address.
The valid format of the address follows this example:
ws://192.168.254.98:8889/gate
When you successfully registered your application to the GDS server, the GDS administrator has to approve your registration. Uploading events to GDS is possible only after approval!
It is recommended to upload your data after finishing the measurement preferably on Wi-Fi connection for the following reasons:

1. As sending images to GDS also uses your 4g connection and it might slow down the response time of the cloud service and it may cause some timeouts when driving in low 4g gradient area.

2. Uploading the overview image on 4g connection is not recommended at all, because an event can be even 2 Mb!! When connected to Wi-Fi, this is not a problem and you can show all data that is generated by the app, on the GDS.

> **!Important**
>
> It is not recommended to touch the phone while uploading, as it will stop the upload procedure.

2.1.3. HTTP POST

Giving a valid HTTP path, the app will make HTTP POST to the given path concatenated with /api/event. The app generates a regular HTTP multipart message with the following content:

field	description	type	example
data	Event JSON	string	{Event JSON…}
image	Binary JPG image of the vehicle	binary JPG	binary data
plate_image	Binary JPG image of the plate	binary JPG	binary data
overview_image	Binary JPG overview image	binary JPG	binary data
video	Binary MP4 video	binary JPG	binary data

An example of a valid path is this: http://192.168.1.50:8081

This is a short HTML that generates the same request by submitting the form:

```
<!DOCTYPE html>
<html>
  <body>
    <form action="http://ip:port/api/event" method="post"
enctype="multipart/form-data">
      <label>data</label>
      <input type="text" name="id" value="{JSON content here…}"
required><br>
      <label>Image</label>
      <input type="file" name="image" accept="image/bmp, image/jpeg,
image/png"><br>
      <label>Plate image</label>
      <input type="file" name="plate_image" accept="image/bmp,
image/jpeg, image/png"><br>
      <label>Overview image</label>
      <input type="file" name="overview_image" accept="image/bmp,
image/jpeg, image/png"><br>
      <label>Video</label>
      <input type="file" name="video" accept="video/mp4"><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

2.1.4. FTP

Giving a valid FTP path, the app will upload the event to the FTP server.

An example of a valid FTP path: ftp://user:password@192.168.43.107:21/

An event creates two or more files in the fileserver side. Images of the vehicle and a valid JSON file with all the metadata. The name of the files are the same, and only the extension is different (JPG, JSON). The name of the files encodes the minimal info about the corresponding event in the following format: nationality_plate_timestamp_latitude_longitude

This is an example:

H_ARH001_1573041317600_47.49485769_19.01995078.jpg

H_ARH001_1573041317600_47.49485769_19.01995078.json

This is an example result of an upload:

2.2. HOTLISTS

Besides storing your hotlists on the device, it is possible to setup a server. The app can send the ANPR results to this server, and receive the hotlist info from the server. In case of a denylisted vehicle, the device makes an alert (visual + sound).

2.2.1. Hotlist server protocol

Request

The android device sends a HTTP POST message to the given url + /api/bwlist path. The app generates a regular HTTP multipart message with the following fields:

- nationality: The plate's nationality
- plate: the plate text

This is an HTML that generates the same message:

```html
<!DOCTYPE html>
<html>
  <body>
    <form action="http://ip:port/api/bwlist" method="post"
enctype="multipart/form-data">
      <label>Plate</label>
      <input type="text" name="plate" value="ARH001"><br>
      <label>Nat</label>
      <input type="text" name="nationality" value="HUN"><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Result

For such a request, the server responds with a JSON text. Currently it has only one field: "type". The value can be "deny", "allow" or "none" (lowercase). An example of such a result is the following:

```
{
      "type":"deny"
}
```

2.2.2. Hotlists from web url

You can setup your hotlists using a web url. The app makes a HTTP GET call to the given link. It expects to reach a CSV file with the following format: nationality;plate;type

This file can be generated by a back-office system. Nationality shall be in ISO3 format.

Here is an example of a valid CSV

```
HUN;KRV914;deny
HUN;MNC878;allow
HUN;PMS314;allow
HUN;NPV421;deny
HUN;RFR826;deny
HUN;LLC481;allow
HUN;LWC739;allow
HUN;LGK007;deny
HUN;MYK030;deny
HUN;HDH062;allow
HUN;KCH452;allow
HUN;LOL247;deny
HUN;PLG012;allow
HUN;HZH004;allow
HUN;EXY124;allow
HUN;POV865;deny
```

2.3. INTEGRATING THE APP INTO YOUR APPLICATION (LEGACY)

2.3.1. Brief how-to

If you have an installed Carmen Mobile app on your device, you can embed it into your application. This section describes the legacy mode. The recommended and more flexible method is described in the 3 chapter if this manual. You can start video processing from your application, and pass the desired settings. The settings are passed as a JSON string. An example of such a JSON is the following:

```json
{
    "startVideo":true,
    "receiverClass":"com.ar.carmenintegration.CarmenResultReceiver",
    "mode":"InMotion"
}
```

Start video processing with the following code in Android:

```java
private void startVideoProcessing() {
    JSONObject settings = new JSONObject();
    try {
        settings.put("startVideo", true);
        settings.put("mode", "InMotion");
        settings.put("receiverClass",
"com.ar.carmenintegration.CarmenResultReceiver");
    } catch (JSONException e) {
        e.printStackTrace();
    }

    Intent launchIntent =
getPackageManager().getLaunchIntentForPackage("com.arh.anprclientwithopencv");

    if (launchIntent != null) {
        launchIntent.putExtra("settings", settings.toString());
        startActivity(launchIntent);
    }
}
```

You can pass the following modes: InMotion, InFixPosition, Parking, InHand, Motorway, Custom.

After starting the app, you have to wait for establishing connection with the cloud. The video processing will be started automatically. The app will send events as Boradcast messages to your caller activity's receiverClass in JSON format.

You will have to create a Broadcast receiver class in your application to receive the events.

Here you can find the details how to create one:

https://developer.android.com/guide/components/broadcasts

2.3.2. Broadcast receiver Examlple

Code

```java
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;

public class CarmenResultReceiver extends BroadcastReceiver {

    public CarmenResultReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        final PendingResult pendingResult = goAsync();
        Task asyncTask = new Task(pendingResult, intent,context);
        asyncTask.execute();
    }
    private static class Task extends AsyncTask<String, Integer, String> {
        private final PendingResult pendingResult;
        private final Intent intent;
        private final Context context;

        private Task(PendingResult pendingResult, Intent intent, Context context) {
            this.pendingResult = pendingResult;
            this.intent = intent;
            this.context = context;
        }

        @Override
        protected String doInBackground(String... strings) {
            if(intent.getAction() != null)
            {
                String json = intent.getStringExtra(handler.getExtra());
                if(json != null)
                {
                    // do whatever you want with your anpr result.
                }
            }
            return "";
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            pendingResult.finish();
        }
    }
}
```

Other important codes & settings

Set up the Manifest file

Add your receiver class

```xml
<receiver android:name="com.ar.carmenintegration.CarmenResultReceiver"
android:exported="true" android:enabled="true">
    <intent-filter>
        <action android:name="com.arh.anprclientwithopencv.EVENT" />
        <action android:name="com.arh.anprclientwithopencv.RESULT" />
    </intent-filter>
</receiver>
```

Add this line in the application tag to be able to access the images as well.

```xml
android:requestLegacyExternalStorage="true"
```

Request the following permissions to get read and write access and to be able to start the app:

```xml
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
```

2.3.3. Anpr Service

When Carmen Mobile is installed on your phone, an anpr service is listening in the background. If your apiKey is set properly and you have access to the Internet, other apps can send images for automatic recognition to CarmenMobile. It will forward the request to the cloud and sends the result to a Broadcast receiver class.

This is a short example of how it is possible to pass an image for automatic recognition.

```java
private void sendBroadcast(String path)
{
    Intent intent = new Intent();
    intent.setAction("READ");
    intent.putExtra("imagePath", path); // path to the image you want to anpr.
    intent.putExtra("id", "your unique id");
    intent.putExtra("receiverClass",
"com.ar.carmenintegration.CarmenResultReceiver");
    intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
    intent.addCategory(Intent.CATEGORY_DEFAULT);
    intent.setComponent(new ComponentName("com.arh.anprclientwithopencv",
"com.arh.anprclientwithopencv.AnprService"));
    context.sendBroadcast(intent);
}
```

Carmen Mobile will get back to you with a RESULT in the receiver class. You can find an example for a receiver in 2.3.2.

3. CARMEN MOBILE API & PLUGIN

From version 1.92 Carmen Mobile has a new way to integrate with third party apps. It opens an API that not only enables integrators to receive events, but it also facilitates communication between external applications and the Carmen Mobile application. It provides a range of functionalities including sending messages, managing hotlists, handling events, controlling specific features, launching external apps, and map interactions.

3.1. HOW IT WORKS

You can write your plugin using Android's broadcast messaging. The app sends you messages about events that occurred, and you can send control messages to Carmen Mobile. You will find examples about how to send and receive messages from and to Carmen Mobile later in this documentation.

3.2. PERMISSIONS

These are the permissions that external apps can request to interact with Carmen Mobile. The user will have to accept these permissions to enable your plugin. All API calls are restricted by these permissions and checked against the requests and callbacks. You will receive events based on the requested permissions, and you can interact with the app only if the user accepted all requested permissions. Permissions can be granted automatically if user passes a valid signature as a registration parameter.

A plugin can request the following permissions:

- MANAGE_EVENTS: Manage events in CarmenMobile. (Delete, update event data.)
- SHOW_MESSAGE: Show messages within CarmenMobile (Use built in messaging.)
- SHOW_DIALOG: Create and display dialog boxes. (Show custom dialogs or use the built in alert)
- MANAGE_HOTLIST: Add to or remove from allow/deny lists.
- UPLOAD_MANAGER: Handle upload-related functionalities.
- MANAGE_FEATURES: Enable or disable specific features.
- OPEN_EXTERNAL_APP: Launch external applications. (You can start your own Activity if needed.)
- LOCATION: Access event location and map functions.

3.3.   START CARMEN MOBILE AND REGISTER YOUR PLUGIN

Extending Carmen Mobile can be done by registering your plugin. This is a sample code that starts

Carmen Mobile with plugin registration.

```java
private void startVideoProcessing() {
    JSONObject settings = new JSONObject();

    try {
        settings.put("startVideo", true);
        settings.put("mode", "InMotion");
        settings.put("registerPlugin", registrationJson.toString());
        settings.put("sdkMode",false);
        settings.put("delayBeforeEventCreation",500);


    } catch (JSONException e) {
        e.printStackTrace();
    }


    Intent launchIntent = new Intent(Intent.ACTION_VIEW);
    launchIntent.setComponent(new ComponentName("com.arh.anprclientwithopencv",
"com.arh.anprclientwithopencv.activities.MainActivity"));


    if (launchIntent != null) {
        launchIntent.putExtra("settings", settings.toString());
        Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.integration);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.PNG, 100, baos);
        byte[] byteArray = baos.toByteArray();

        launchIntent.putExtra("icon", byteArray);
        startActivity(launchIntent);
    }
    else {
        Log.e(TAG, "intent null");
    }
}
```

3.3.1. Configuration setup

"`startVideo`": true - Indicates that video processing should start.

"`mode`": "InMotion" - Specifies the video processing mode. It can be "InMotion", "Parking",

"Handheld", "InFixPosition", "AboveMotorway"

"`receiverClass`": "com.ar.carmenintegration.CarmenResultReceiver" - Defines the class that

will receive the results from the video processing. This is the legacy integration method described in

2.3 section of this document.

"`registerPlugin`": registrationJson.toString() - Adds the plugin registration details, which have

been previously set in registrationJson an example can be found in an upcoming section.

"`sdkMode`": false - Specifies that the application should not run in SDK mode. If set to true users will

not be able to open CarmenMobile unless they start your app first. SDK mode is enabled only in case

of valid signature.

"`delayBeforeEventCreation`": 500 - Introduces a delay (in milliseconds) before the event

creation. Plugins have a beforeEventCreation callback. After it is called the app waits for this amount

of time (milliseconds) for plugins to be able call their own logic. This makes possible to put event to

allowlist or disable upload when plugin specified conditions hold.

3.3.2. Register plugin settings

The following line is needed for plugin registration:

settings.put("registerPlugin", registrationJson.toString());

You must construct the above json object for plugin registration with the following details:
A JSON object containing the following details:

- o name: The name of the plugin.
- o package: The package name where the plugin resides.
- o class: The full canonical name of the class implementing the the BroadcastReceiver that receives the messages from CarmenMobile.
- o signature: A unique cryptographic signature that represents the plugin. This can be requested from Adaptive Recognition. This is crucial for the platform's verification process. If not given or not valid, plugin can still register, but users will be notified, that an unknown source is trying to get control and info. The user has to accept the permissions as well.
- o permissions: A list of permissions that the plugin requires to operate correctly. This must be a JSONArray of strings with capital lettered permissions like: "MANAGE_EVENTS"

```java
JSONObject registrationParameters = new JSONObject();
try {
    registrationParameters.put("name", plugin.getName());
    registrationParameters.put("package", "com.ar.carmenintegration");
    registrationParameters.put("class", getClass().getCanonicalName());
    registrationParameters.put("signature", "YOUR_SIGNATURE");
    JSONArray permissions = new JSONArray();
    for (String permission : plugin.getRequiredPermissions() ) {
        permissions.put(permission);
    }
    registrationParameters.put("permissions", permissions);
} catch (JSONException e) {
    e.printStackTrace();
}
```

Adaptive Recognition **America**
Adaptive Recognition **Hungary**
Adaptive Recognition **Nordic**
Adaptive Recognition **Singapore**
REQUESTINFO@ADAPTIVERECOGNITION.COM
WWW.ADAPTIVERECOGNITION.COM

3.4. API METHODS

3.4.1. How to construct the commands.

Each command can be called with an Intent sent as Broadcast message.

```java
public void addPlateToAllowList(String country, String plate) {
    String action = "addPlateToAllowList";
    JSONObject parameters = new JSONObject();
    try {
        parameters.put("country", country);
        parameters.put("plate", plate);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    sendBroadcast(action, parameters.toString());
}
```

In this example you can see that the action is always the name of the function, and the parameters of the function goes to a JSON object called parameters. Then a broadcast message is sent with this content. Let us see how a broadcast message is sent;

```java
private void sendBroadcast(String action, String parameters)
{
    Intent intent = new Intent();
    intent.setAction("com.arh.anprclientwithopencv.CONTROLMESSAGE");
    intent.putExtra("action", action);
    intent.putExtra("parameters", parameters);
    intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
    intent.addCategory(Intent.CATEGORY_DEFAULT);
    intent.setComponent(new
ComponentName("com.arh.anprclientwithopencv","com.arh.anprclientwithopencv.Controller"));
    Log.i(TAG,"Broadcast message sent");
    context.sendBroadcast(intent);
}
```

In a nutshell, you must construct an intent and send it as a broadcast message, targeting com.arh.anprclientwithopencv.Controller class. The intent action must be com.arh.anprclientwithopencv.CONTROLMESSAGE and two parameters are necessary. These are a string parameter called action, which is the functionality you want to call (name of the function in the upcoming sections), and the parameters, which is a JSON encoded as string, that represents the parameters (function parameters in the upcoming section), that are needed to perform the required action. If a function needs no parameters, just leave it empty.

3.4.2. Messages and Alerts

Needs „SHOW_MESSAGE and/or SHOW_DIALOG" permission

- `createMessage(String type, String message, long duration)`: Display a message with a specific type for a defined duration. String type parameter can be one of the following: normal, warning, error. Calling this functionality will create a message like the example below.
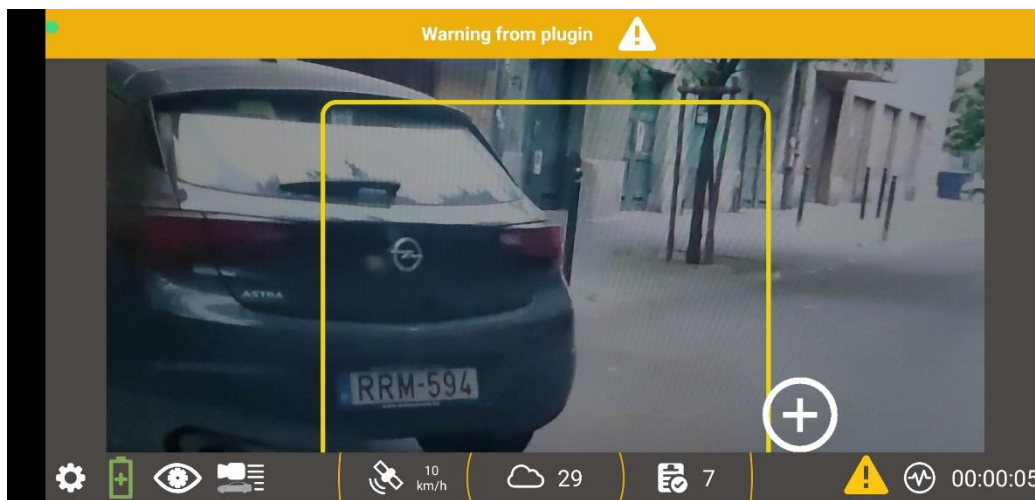
- `createEventAlert(String eventJSON, String message, int colorCode)`: Create an event alert with a specified message and color. In this case the eventJSON must be the same that you receive in beforeEventCreation or afterEventCreation callbacks (discussed in chapter 3.6). Calling event alert will create the same type of alert, that Carmen Mobile generates for deny listed vehicles. In the example below, the plugin generates alert for all German registration plates.
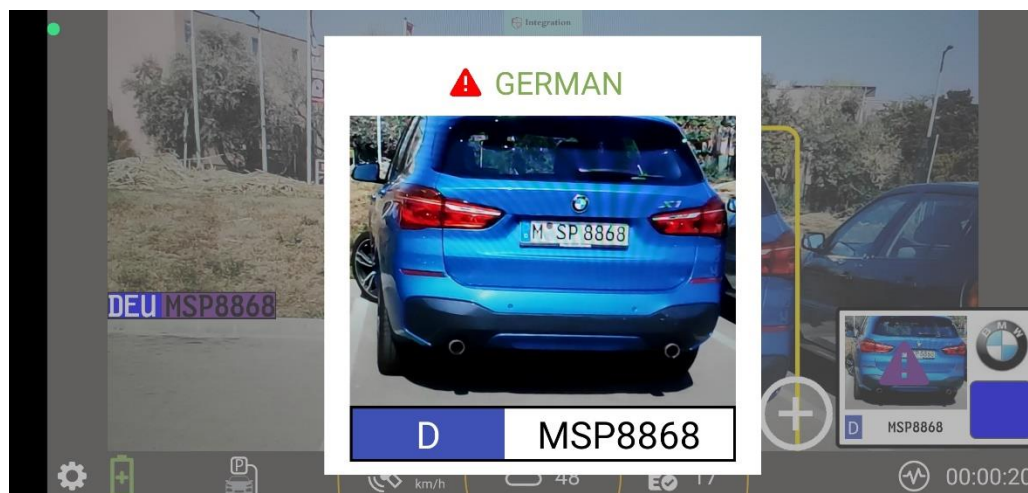
- `createDialog(String dialogDescriptor)`: Display a dialog based on the provided descriptor. Dialog descriptor is a json string that describes a dialog. Detailed description about it can be found in this documentation at 3.5 section.
- `toggleActionButtonVisibility(boolean visibility)`: Show or hide the action button in CameraActivity. The action button can be found in the right bottom corner.

3.4.3. Hotlist Management

Needs „MANAGE_HOTLIST" permission

- `addPlateToAllowList(String country, String plate):` Add a plate from a specific country to the allow list.
- `addPlateToDenyList(String country, String plate):` Add a plate from a specific country to the deny list.
- `removePlateFromHotlist(String country, String plate):` Remove a plate from the hotlist.
- `clearLists():` Clear all entries from the allow and deny lists.

- `void highlightEventWithColor(String country, String plate, int colorCode, boolean attachToEvent):` Highlights an event based on country and plate with a given color. If attachToEvent is true the app also puts the highlight color in the metadata of the event. colorCode is an integer with the following encoding. 0 – RED, 1 – GREEN, 2-PURPLE, 3-BLUE,4-YELLOW. This functionality needs „MANAGE_EVENTS" permission as well. In the next example the plugin assigns random colors to events. Both the plate and the event in the right slider will be highlighted with the specified color.
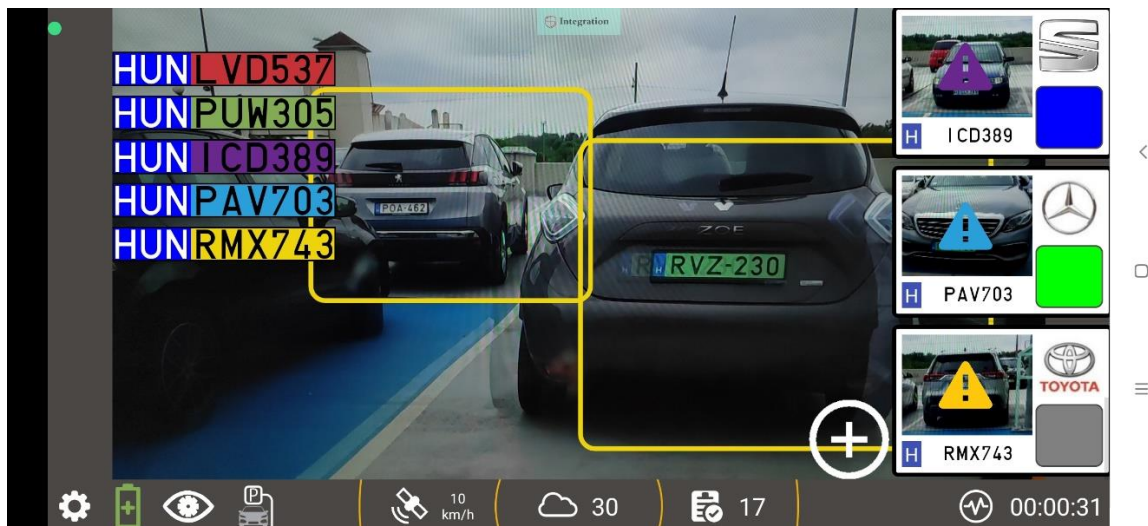
3.4.4. Event Handling

Needs „MANAGE_EVENTS" permission.

- `deleteEvent(String plate, long timestamp):` Delete a specific event based on plate and timestamp.
- `deleteAllEvents():` Remove all events.

- `attachMetadataToEvent(String plate, long timestamp, String key, String json):` Add metadata to a specific event.
- `mergeMetadataIntoEvent(String plate, long timestamp, String json):` Merge metadata into an existing event.

> **! Important**
>
> This method can overwrite elements in the original event data. Use unique keys in your json!

3.4.5. Feature Management

Needs „MANAGE_FEATURES" permission.
- enableFeature(String feature): Activate a specific feature.
- disableFeature(String feature): Deactivate a specific feature.

These are the features that can be enabled or disabled within CarmenMobile.

- VEHICLE_TRIGGER: automatic trigger
- UPLOAD: Event upload
- DECORATIONS_SPIRITLEVEL: spirit level
- DECORATIONS_PLATES: up sliding license plates in camera mode
- OFFLINE_DATA_COLLECTION: Gather data without internet connectivity.
- DATA_OVERVIEW_IMAGE: Save overview image.
- DATA_MMR: Read make and model and color of event.
- OPTIMIZATIONS_POWERSAVING: Powersaving mode, when enabled fewer images are processed.
- OPTIMIZATIONS_GRAYSCALE: Processing only grayscale images saves power.
- OPTIMIZATIONS_CREDITSAVING: uses less credit, but can cause delay in event processing.

This is an example how to enable make and model recognition from your application.

```java
String action = "enableFeature"; // disabling can be done with:
disableFeature
JSONObject parameters = new JSONObject();
try {
    parameters.put("feature","DATA_MMR");
} catch (JSONException e) {
    e.printStackTrace();
}


//Then use sendBroadcast method to send your request detailed in section
3.4.1
sendBroadcast(action, parameters.toString());
```

3.4.6. External App Handling

Needs „OPEN_EXTERNAL_APP" permission.
- openExternalApp(String packageName, String className, String intentDataJson): Launch an external app using package and class names, along with intent data. If intentDataJson holds an event, the image is also passed in the launch intent. This is an easy way to call back to your own application. An example of this use case can be when a certain event needs complex logic, like fining a vehicle. Then you can do all the necessary coding in your own Activity. If your logic is done you can go back to Carmen Mobile for further event collection, by clicking the back button on the device.

3.4.7. Map Handling

Needs „SHOW_DIALOG & LOCATION" permission. Map can be used only in CameraActivity, so in case it is needed, you must wait for the onResume callback with "CameraActivity" parameter.

Fist you must call the openMap method. After you received the mapReady callback from Carmen Mobile, you can place markers and events on the map. The map will not automatically follow the latest event, so in order to do so, you also have to call the centerMapToCoordinates method with the desired location.

- `openMap(double latitude, double longitude, int zoomLevel):` Open the map centered on a specific location and zoom level.
- `closeMap():` Close the map view.

- `centerMapToCoordinates(double latitude, double longitude, float zoom):` Center the map on a location with a specified zoom.
- `addMarker(String id, double latitude, double longitude, String title, String snippet):` Add a map marker with specific details.
- `addVehicleMarker(String id, String eventJSON, long ttl):` Add a vehicle marker with an associated event and time-to-live.
- `removeMarker(String id):` Delete a marker based on its ID.
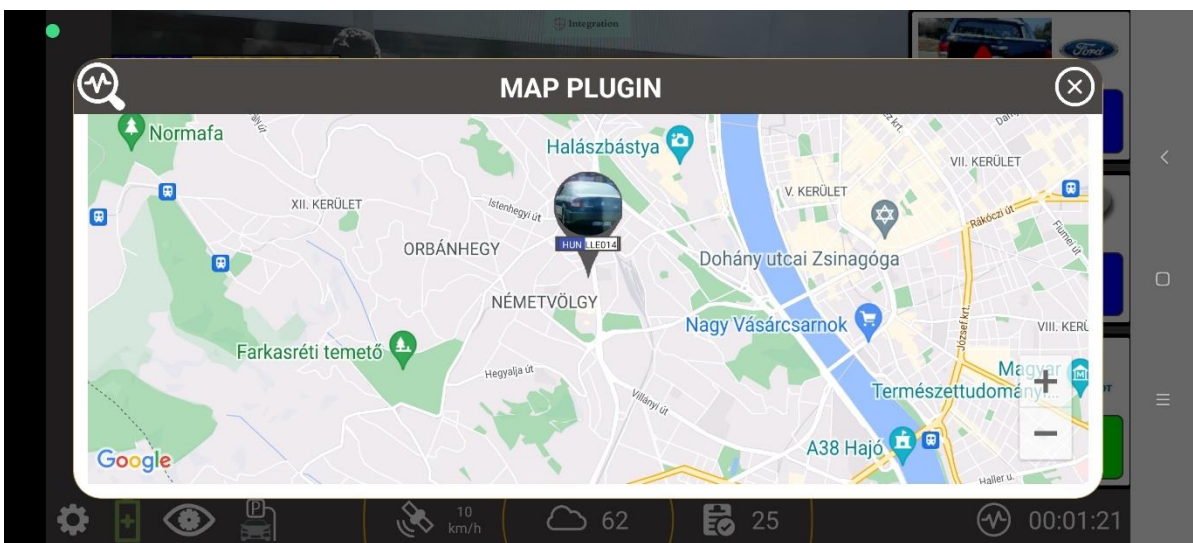
After placing markers with id, you will get callback message (onMarkerClick), in case a user clicks on one of them.

3.4.8. Upload Handling

Needs „UPLOAD_MANAGER" permission

- `disableUpload(String plate, long timestamp):` Disable upload for a specific event based on plate and timestamp. For this function to operate, you need to set the "delayBeforeEventCreation" in the start command. After receiving the event in the beforeEventCreation callback you have that time to do your own logic to decide if the event upload needs to be disabled. If the delay is passed disabling upload will not work.

3.5. DIALOG DESCRIPTOR JSON DOCUMENTATION

3.5.1. Overview

The dialog descriptor JSON provides a structured way to describe a UI dialog, including its layout and individual components. By following the structure, you can define a custom dialog that will be rendered accordingly.

3.5.2. JSON Structure

views: A list containing one or more view objects. Each view object represents a container or a widget.
Each view object can have the following properties:

- type: Indicates the type of view or widget. The supported types with configurable properties are the following:
    o LinearLayout (id, width, height, orientation, layout_weight, children)
    o Button (id, width, height, text)
    o TextView (id, width, height, text)
    o EditText (id, width, height, text)
    o RadioGroup (id, width, height)
    o RadioButton (id, text)
    o Switch (id, text, checked)
    o NumberPicker (id, width, height, min, max, default)
- id: A unique identifier for the view. It's useful for referencing and manipulating the view programmatically. This is the id that your plugin callback function will receive. For example if button is clicked in your dialog, the "buttonClicked(String id)" callback will receive clicked button's id.
- text (optional): The display text for the widget, where applicable. For instance, for TextView, Button, and Switch.
- checked (optional): A boolean that indicates whether a Switch or RadioButton is on (true) or off (false).
- orientation (only for LinearLayout): Describes the arrangement direction of its children. Can be vertical or horizontal.
- width (optional): Specifies the width of the view. Values can be:
    o match_parent: The view will expand to match the width of its parent container.
    o wrap_content: The view will size itself to fit its content.
- height(optional): Specifies the height of the view.
- layout_weight (specific to LinearLayout when in horizontal or vertical orientation): This determines how much of the remaining space in the layout should be allocated to the view. It's a proportion value that decides how space is distributed among sibling views in a LinearLayout.
- children (only for LinearLayout): A list of child view objects nested inside the layout.

3.5.3. Usage Example

Let us construct a dialog with a vertical LinearLayout containing two switches:

```
{
  "views": [
    {
      "type": "LinearLayout",
      "id": "layout_id",
      "orientation": "vertical",
      "children": [
        {
          "type": "Switch",
          "id": "switch1_id",
          "text": "Switch 1 Label",
          "checked": true
        },
        {
          "type": "Switch",
          "id": "switch2_id",
          "text": "Switch 2 Label",
          "checked": false
        }
      ]
    }
  ]
}
```

This image illustrates how the dialog will appear on the screen.
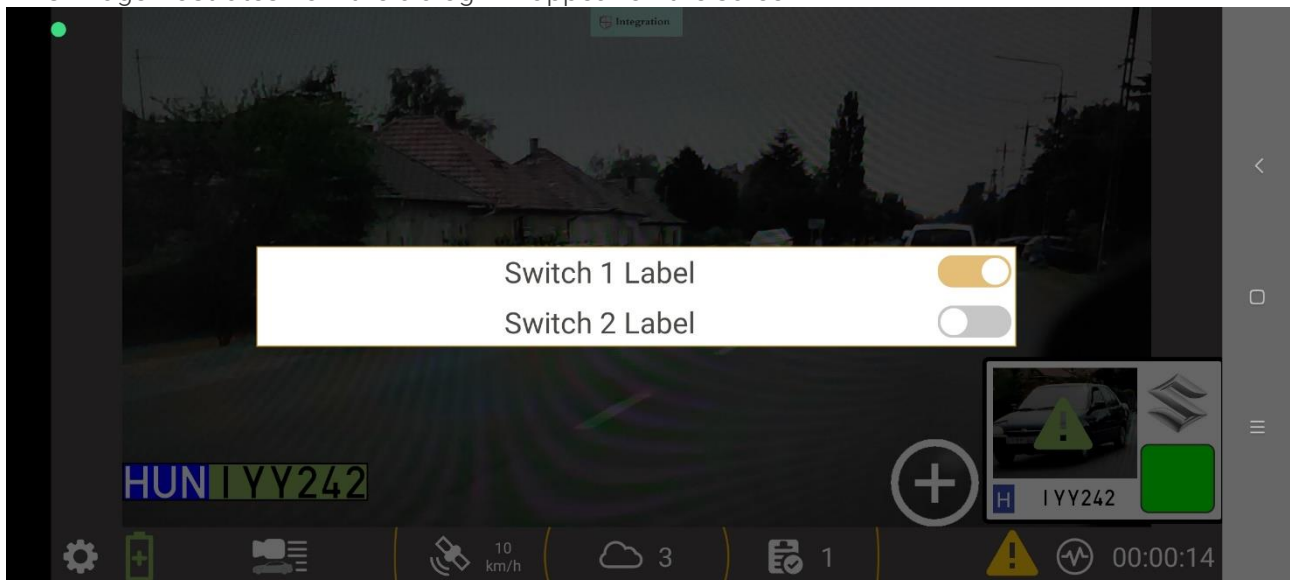
3.6.   PLUGIN CALLBACKS

As Carmen Mobile sends broadcast messages to your app, you need to create a class, that listens to these Control messages. Do not forget to declare this class in the AndroidManifest as well like this:

```xml
<receiver
    android:name=".controller.CarmenMobileController"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="com.arh.anprclientwithopencv.CONTROLMESSAGE" />
    </intent-filter>
</receiver>
```

An example of such a receiver can be found in the appendix (4.1) of this document.

From the plugin side you receive info about the following events.

```java
void beforeEventCreation(String eventJSON);
// called when new event created, but before archiveing & uploading.
void afterEventCreation(String eventJSON);
// called after all processing / uploading done regarding to the event.
void onEventClick(String eventJSON);
// called when user clicks on an event in camera mode.
void onActionButtonClicked();
// called when user clicks on the action button in camera mode.
void onActivityCreated(String activityName);
// called when activity created.
void onActivityDestroyed(String activityName);
// called when activity destroyed, user clicked back button.
void onActivityResumed(String activityName);
// called when activity resumed
void onActivityPaused(String activityName);
// called when activity paused.
void buttonClicked(String id);
// called when a button with id clicked by the user.
void switchChanged(String id, boolean isChecked);
// called when a switch state is changed by the user.
void numberPickerChanged(String id, int newVal);
// called when a number is picked by the user.
void radioButtonChanged(String id, boolean isChecked);
// called when a radio button clicked by the user.
void onMapReady();
// called when a map is ready for placing events/ markers on it.
void onMapClosed();
// called when the map is closed and no longer available for placing markers.
void onMarkerClick(String markerId, String title, String snippet);
// called when the user clicks on a marker in the map.
```

These functions are called as Broadacst messages. An example class that delivers you these callback functions can be found in the appendix: 4.1

3.7.   ICON

If you have a valid signature form Adaptive Recognition, you can replace the Adaptive Recognition logo to your own logo. Together with the sdk mode these functions provide a white labelled mode.

3.8.   FILE SERVICE

Carmen Mobile starts a service for easy file access on the localhost:8080.

You can retrieve image by making a HTTP GET with the filename. This is an example GET method.

http://localhost:8080/api/image/car_JES888_169637927489_1993922552316.jpg

You can retrieve both main and overview images. Images can be retrieved until the app is running.

4.   APPENDIX

4.1.   CARMENMOBILECONTROLLER EXAMPLE

```java
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class CarmenMobileController  extends BroadcastReceiver {
    Context context = null;
    String currentActivity = "";
    private String TAG = CarmenMobileController.class.getSimpleName();


    public void setContext(Context c)
    {
        this.context = c;
    }

    public CarmenMobileController() {

    }

    @Override
    public void onReceive(Context context, Intent intent) {
        setContext(context);
        String action = intent.getStringExtra("action");
        Log.i(TAG, "onReceive: " + action);

        if(action.equals("beforeEventCreation"))
        {
            String eventJSON = intent.getStringExtra("eventJSON");
        }
        if(action.equals("afterEventCreation"))
        {
            String eventJSON = intent.getStringExtra("eventJSON");
        }
        if(action.equals("onTextChanged"))
        {
            String id = intent.getStringExtra("id");
            String text = intent.getStringExtra("text");
        }
        if(action.equals("onActivityCreated"))
        {
            String activityName = intent.getStringExtra("activityName");
        }
        if(action.equals("onActivityResumed"))
        {
            currentActivity = intent.getStringExtra("activityName");
        }
        if(action.equals("onActivityPaused"))
        {
            String activityName = intent.getStringExtra("activityName");
        }
        if(action.equals("onActivityDestroyed"))
```

Adaptive Recognition **America**
Adaptive Recognition **Hungary**
Adaptive Recognition **Nordic**
Adaptive Recognition **Singapore**
REQUESTINFO@ADAPTIVERECOGNITION.COM
WWW.ADAPTIVERECOGNITION.COM

```java
        {
            String activityName = intent.getStringExtra("activityName");
        }
        if(action.equals("onEventClick"))
        {
            String eventJSON = intent.getStringExtra("eventJSON");
        }
        if(action.equals("onMapClosed"))
        {
        }
        if(action.equals("onMapReady"))
        {
        }
        if(action.equals("buttonClicked"))
        {
            String id = intent.getStringExtra("id");
        }
        if(action.equals("switchChanged"))
        {
            String id = intent.getStringExtra("id");
            boolean value = intent.getBooleanExtra("value", false);
        }
        if(action.equals("numberPickerChanged"))
        {
            String id = intent.getStringExtra("id");
            int value = intent.getIntExtra("value", 0);
        }
        if(action.equals("radioButtonChanged"))
        {
            String id = intent.getStringExtra("id");
            boolean value = intent.getBooleanExtra("value", false);
        }
        if(action.equals("onActionButtonClicked"))
        {
        }
        if(action.equals("onMarkerClick"))
        {
            String markerId = intent.getStringExtra("markerId");
            String title = intent.getStringExtra("title");
            String snippet = intent.getStringExtra("snippet");
        }
    }


    @Override
    public void createMessage(String type, String message, long duration) {
        String action = "createMessage";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("type", type);
            parameters.put("message", message);
            parameters.put("duration", duration);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void createEventAlert(String eventJSON, String message, int
colorCode) {
        String action = "createEventAlert";
        JSONObject parameters = new JSONObject();
```

```java
        try {
            parameters.put("eventJSON", eventJSON);
            parameters.put("message", message);
            parameters.put("colorCode", colorCode);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void createDialog(String dialogDescriptor) {
        String action = "createDialog";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("dialogDescriptor", dialogDescriptor);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void toggleActionButtonVisibility(boolean show) {
        String action = "toggleActionButtonVisibility";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("visibility", show);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void addPlateToAllowList(String country, String plate) {
        String action = "addPlateToAllowList";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("country", country);
            parameters.put("plate", plate);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void removePlateFromHotlist(String country, String plate) {
        String action = "removePlateFromHotlist";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("country", country);
            parameters.put("plate", plate);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void addPlateToDenyList(String country, String plate) {
```

```java
        String action = "addPlateToDenyList";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("country", country);
            parameters.put("plate", plate);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void clearLists() {
        String action = "clearLists";
        JSONObject parameters = new JSONObject();
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void highlightEventWithColor(String country, String  plate, int
colorCode, boolean attachToEvent) {
        String action = "highlightEventWithColor";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("country", country);
            parameters.put("plate", plate);
            parameters.put("colorCode", colorCode);
            parameters.put("attachToEvent", attachToEvent);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }


    @Override
    public void deleteEvent(String plate, long timestamp) {
        String action = "deleteEvent";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("plate", plate);
            parameters.put("timestamp", timestamp);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void deleteAllEvents() {
        String action = "deleteAllEvents";
        JSONObject parameters = new JSONObject();
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void attachMetadataToEvent(String plate, long timestamp, String key,
String json) {
        String action = "attachMetadataToEvent";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("plate", plate);
            parameters.put("timestamp", timestamp);
```

```java
                parameters.put("key", key);
                parameters.put("json", json);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            sendBroadcast(action, parameters.toString());
        }


    @Override
    public void mergeMetadataIntoEvent(String plate, long timestamp, String
json) {
            String action = "mergeMetadataIntoEvent";
            JSONObject parameters = new JSONObject();
            try {
                parameters.put("plate", plate);
                parameters.put("timestamp", timestamp);
                parameters.put("json", json);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            sendBroadcast(action, parameters.toString());
        }


    @Override
    public void enableFeature(String feature) {
            String action = "enableFeature";
            JSONObject parameters = new JSONObject();
            try {
                parameters.put("feature", feature);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            sendBroadcast(action, parameters.toString());
        }


    @Override
    public void disableFeature(String feature) {
            String action = "disableFeature";
            JSONObject parameters = new JSONObject();
            try {
                parameters.put("feature", feature);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            sendBroadcast(action, parameters.toString());
        }


    @Override
    public void openExternalApp(String packageName, String className, String
intentDataJson ) {
            String action = "openExternalApp";
            JSONObject parameters = new JSONObject();
            try {
                parameters.put("packageName", packageName);
                parameters.put("className", className);
                parameters.put("intentData", intentDataJson );
            } catch (JSONException e) {
                e.printStackTrace();
            }
            sendBroadcast(action, parameters.toString());
        }


    @Override
```

```java
    public void openMap(double latitude, double longitude, int zoomLevel) {
        String action = "openMap";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("latitude", latitude);
            parameters.put("longitude", longitude);
            parameters.put("zoom", zoomLevel);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void closeMap() {
        String action = "closeMap";
        JSONObject parameters = new JSONObject();
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void centerMapToCoordinates(double latitude, double longitude, float
zoom) {
        String action = "centerMapToCoordinates";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("latitude", latitude);
            parameters.put("longitude", longitude);
            parameters.put("zoom", zoom);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void addMarker(String id, double latitude, double longitude, String
title, String snippet) {
        String action = "addMarker";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("id", id);
            parameters.put("latitude", latitude);
            parameters.put("longitude", longitude);
            parameters.put("title", title);
            parameters.put("snippet", snippet);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }


    @Override
    public void addVehicleMarker(String id, String eventJSON, long ttl) {
        String action = "addVehicleMarker";

        JSONObject parameters = new JSONObject();
        try {
            JSONObject event = new JSONObject(eventJSON);
            parameters.put("id", id);
            parameters.put("eventJSON", event);
            parameters.put("ttl", ttl);
```

```java
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void removeMarker(String id) {
        String action = "removeMarker";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("id", id);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }

    @Override
    public void disableUpload(String plate, long timestamp) {
        String action = "disableUpload";
        JSONObject parameters = new JSONObject();
        try {
            parameters.put("plate", plate);
            parameters.put("timestamp", timestamp);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        sendBroadcast(action, parameters.toString());
    }
```

```java
    public JSONObject getRegistrationCommand()
    {
        JSONObject registrationParameters = new JSONObject();
        try {
            registrationParameters.put("name","YOUR PLUGIN NAME");
            registrationParameters.put("package", "com.your.package");
            registrationParameters.put("class", getClass().getCanonicalName());
            registrationParameters.put("signature", "YOUR KEY");
            JSONArray permissions = new JSONArray();
            for (String permission : plugin.getRequiredPermissions() ) {
                permissions.put(permission);
            }
            registrationParameters.put("permissions", permissions); // JSON
Array of permissions needed by your app.
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return registrationParameters;
    }


    private void sendBroadcast(String action, String parameters)
    {
        Intent intent = new Intent();
        intent.setAction("com.arh.anprclientwithopencv.CONTROLMESSAGE");
        intent.putExtra("action", action);
        intent.putExtra("parameters", parameters);
        intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
        intent.addCategory(Intent.CATEGORY_DEFAULT);
        intent.setComponent(new
ComponentName("com.arh.anprclientwithopencv","com.arh.anprclientwithopencv.Contr
oller"));
        Log.i(TAG,"Broadcast message sent");
        context.sendBroadcast(intent);
    }

}
```

ADAPTIVE RECOGNITION

Contact Information

Headquarters:
Adaptive Recognition Hungary Inc.
Alkotás utca 41 HU-
1123 Budapest Hungary
Phone: +36 1 201 9650
Fax: +36 1 201 9651

Web: www.adaptiverecognition.com

Service Address:
Adaptive Recognition Hungary Inc.
Ipari Park HRSZ1113/1 HU
2074 Perbál Hungary
Phone: +36 1 2019650
E-mail: rmarequest@adaptiverecognition.com

Adaptive Recognition Hungary Technical Support System (ATSS) is designed to provide you the fastest and most proficient assistance, so you can quickly get back to business.
For further technical information about our products, please visit our official website.

Information regarding hardware, software, manuals, and FAQ are easily accessible for customers who previously registered to enter the dedicated ATSS site. Besides offering assistance, the site is also designed to provide maximum protection while managing your business information and the technical solutions utilized.

New User
If this is your first online support request, please create an account by clicking on this link.

Returning User
All registered ATSS customers receive a personal access link via e-mail. If you previously received a confirmation message from ATSS, it contains the embedded link that allows you to securely enter the support site.

If you need assistance with login or registration, please contact atsshelp@adaptiverecognition.com for help.