About This Document

This document is a technical article containing information on how to use some of the Google APIs in Android apps. To understand the contents of this article, one should know the basics of Java and Android app development. Knowledge of the aforementioned APIs is not required. Additionally, the assumption that the developer uses Eclipse IDE was made. The article will guide you through the process of creating your first app making use of Google Drive and Google+ APIs. It also provides general information about Google APIs.

1. Introduction to using Google APIs in Android

1.1 Introduction

Google offers many different services e.g. Drive, Plus, Maps, YouTube and so on. The significant percentage of them has dedicated API which allows using them programmatically from within the apps. This article presents how to use Google Drive and Google+ APIs directly from the Android app. The first thing you have to do, regardless the target platform, is creating API Project in Google APIs Console, which is available under the following link:https://code.google.com/apis/console/. The Console is the place where you turn on and off access to particular API. Each of the APIs has daily courtesy limit which cannot be exceeded by your app. This is the time to turn the Drive API, Drive SDK and Google+ API on. The next thing is creating the Client ID in the API Access tab. The Client ID defines app that has authorized API access and therefore is allowed to communicate with the Google's server on your behalf. It is the crucial part of the OAuth 2.0 authorization protocol, which is becoming de facto a standard for the API authorization. While creating the Client ID for the mobile app there are 3 things you have to provide. First of all you have to set the Application type to Installed application. Next things is providing package name of your Android app. The last thing is entering signing certificate fingerprint, which identifies the certificate used to sign the Android's apk file. Such certificate fingerprint can be easily generated with the keytool which is a part of Java's toolkit. It is important that the package name and the fingerprint provided in the console match the package name and the fingerprint of your app. If this condition is not met, the Google's server will not authorize your app and as a result you will not be able to access any of the APIs. Having configured the API project with the Client ID, you could move on to creating the Android project with Google libraries.

1.2 Setting up Android project for using Google APIs

The first, obvious step you should take is creating a new Android project or opening existing one if you have any. Now you need to add the Google's libraries to the project to be able to start developing code which integrates the Google's APIs with your app. Until recently, it was necessary to manually download all needed libraries, but fortunately for the developers things changed. There are to libraries you need: the Google Play Services library, which handles authorization process and the library for the specific API. The Google Play Services library can be obtained with the Android SDK Manager. It will be stored in the extras folder of the Android SDK. You have to add it to your project's Build Path as External JAR. It is also recommended to mark this JAR as exported. The second library you need is the library for the specific Google's API. Thanks to Google's developers you don't have to look for this library manually. All you have to do is to install the Google Plugin for Eclipse (available under the following link: https://developers.google.com/eclipse/). Then you can right click on your project, select Google->Add Google APIs... and choose the API you would like to use in your app. The plug-in will automatically download all the necessary JARs and add them to the project's Build Path. Now you are ready to start the development of the integration with the chosen Google API.

1.3 Handling the authorization process

Regardless the API you decided to use, there is one common thing that has to be always done before

sending requests to the Google's servers - authorization process. Fortunately Google provides the Google Play Services library which handles most of the job related to the authentication. Thanks to it you almost do not have to directly interact with OAuth 2.0 authorization flow. The code snippet for handling the authorization process is presented below. All the irrelevant details (import statements, not important attributes) were omitted.

```
public class MainActivity extends Activity {

    private TextView output;
    private GoogleAccountCredential credential;
    private static final int CHOOSE_ACCOUNT = 1;
    private static final int REQUEST_AUTHORIZATION = 2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        output = (TextView) findViewById(R.id.output);

        credential = GoogleAccountCredential.usingOAuth2(this, DriveScopes.DRIVE,
PlusScopes.PLUS_ME);
        startActivityForResult(credential.newChooseAccountIntent(), CHOOSE_ACCOUNT);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch(requestCode) {
            case CHOOSE_ACCOUNT:
                if(data!=null) {
                    Bundle extras = data.getExtras();
                    if (extras!=null) {
                        String accountName =
extras.getString(AccountManager.KEY_ACCOUNT_NAME);

                        if(accountName != null) {
                          credential.setSelectedAccountName(accountName);
                        }
                    }
                }
                break;
            case REQUEST_AUTHORIZATION:
                if (resultCode == Activity.RESULT_OK) {
                    Toast.makeText(this, "Click the button again to perform chosen
```

```
action.", Toast.LENGTH_LONG).show();
                } else {
                    startActivityForResult(credential.newChooseAccountIntent(),
CHOOSE_ACCOUNT);
                }
                break;
        }
    }
}
```

[Code 1] Handling the authorization process

The authentication process is fairly easy. The first thing you have to do in the onCreate method is create the GoogleAccountCredential object using the usingOAuth2 static method. This method takes as a constructor's parameters context and String vararg which indicates the scope of the API access. In case of the presented sample app the scopes are DriveScopes.DRIVE and PlusScopes.PLUS_ME. You have to declare that the scopes to have access to specific information from the APIs. The list of the scopes and their detailed descriptions can be found in the documentation of each API. When the credential object is ready, you have to start the activity for the result, using the Intent provided by the newChooseAccountIntent() method from the GoogleAccountCredential's. Apart from passing the Intent to the startActivityForResult method, you also pass the request code which in our case is stored in the CHOOSE_ACCOUNT static variable. This request code is needed to properly handle the response from the started activity. As a result, a new activity with Google account picker is opened. The user is asked to pick one of his/her accounts. When they do, the application flow goes back to MainActivity by the means of the onActivityResult method. This method is the place where the rest of the authentication process is performed. At the beginning of this method the results of various activities are filtered based on the request code. In case the request code is equal to CHOOSE_ACCOUNT, the data attached to activity's result is retrieved with the getExtras() method and stored in the extras Bundle. One element of the information contained in that Bundle is the user's account name. This account name has to be retrieved and set in the GoogleAccountCredential instance with the help of the setSelectedAccountName(String name) method, which is the end of the authentication process. The next step is the creation of the object for communication with the selected APIs. In the case of the presented sample app, there are created objects of Drive and Plus classes. More detailed information about this is presented in the subsequent sections of this article.

2. Using the Google Drive API

2.1 Getting started

Google Drive is Google's cloud-based file storage service, which allows users to store up to 5GB of data for free. It can be accessed from an Android device in two ways: using the native Google Drive app or communicating with Google's server through the Google Drive API directly from your app. The latter case is a very good way to enrich your mobile app with the file storage capability and thus make it more cloud-oriented.

The Google Drive API allows you to perform the following actions:

- Uploading/Downloading files to/from the Drive
- Creating new directories

- Sharing files with the other Drive's users
- Setting access permissions for each file

The most convenient way to communicate with the Drive API is using the Google Drive SDK for Android which contains all the necessary libraries and takes the burden of manually handling HTTP communication. It is also possible to manually construct and send the request to Google's servers but such an approach is very laborious and should be avoided. Before delving into the code which interacts with the Drive you should make sure that your app's user has choosen their Google account and you have the instance of the GoogleAccountCredential class with the proper account name set. Additionally, you need to add to your project the Drive API libraries that can be easily achieved using the Google Plugin for Eclipse. Both things were described in the greater detail in the Introduction section. It is also necessary to enable the Google Drive API and Google Drive SDK in Google's API Console. Otherwise the Google server will reject all the requests.

The last thing which prevents you from accessing the Drive API's functionalities is lack of the Drive object which is the starting point for using the API. This should be created as per the following code snippet:

```
Drive drive = new Drive.Builder(AndroidHttp.newCompatibleTransport(), new GsonFactory(),
credential).build();
```

[Code 2] Creating Drive instance

With the app attached to this article, the Drive instance is created in the onActivityResult method after the user has chosen the Google account to use with the app. The credential object used as the third parameter in Drive.Builder's constructor is the instance of the GoogleCredentialManager class with the account name set. The following subsections present how to utilize a Drive object to communicate with the Drive API.

2.2 Listing files from Drive

The first thing to be presented is a list of files which are stored on the Drive. It is fairly easy and requires only one call to the Drive API. Below is the code snippet which performs these actions. This code is invoked each time user presses the 'Get list of files from Drive account' button.

```
Thread t = new Thread(new Runnable() {

            @Override
            public void run() {
                try {
                    final StringBuilder sb = new StringBuilder();
                    FileList list = drive.files().list().execute();
                    List<File> items = list.getItems();
                    for (File f : items) {
                        sb.append(f.getTitle()).append("\n");
                    }

                    runOnUiThread(new Runnable() {
```

```
                        @Override
                        public void run() {
                            output.setText(sb.toString());
                            Toast.makeText(MainActivity.this, "Files list successfully
downloaded", Toast.LENGTH_LONG).show();
                        }

                    });

            } catch (UserRecoverableAuthIOException e) {
                startActivityForResult(e.getIntent(), REQUEST_AUTHORIZATION);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    });
    t.start();
```

[Code 3] Snippet of the MainActivity.java responsible for listing files stored on the Google Drive

The first thing to be noticed is that since all Drive requests are blocking, they should be invoked in a separate thread. Therefore a new thread is spawned each time the user presses the button responsible for invoking the described code. At the beginning of the try block StringBuilder is created, which is used later to build the String containing a list of all the files. The next line is the place where the magic happens. The first method to be invoked is the files() method which returns a Files object. This object provides methods for the various file operations e.g. inserting a new file or deleting an existing one. In this example the list() method is used, which returns a List request object. At this point nothing interesting happens as there is one last thing lacking. The execute() method has to be invoked on the List object. Invokation of this method starts blocking network operation, which communicates with the Drive API and gets all the necessary information. As a result the FileList object is returned which contains the information about all the files stored on the Drive. The next few lines simply get all the items from the FileList using getItems(), iterates over all of them and adds the title of each File item to the StringBuilder. There is one thing worth noting, the File object contained is not the same File object that is used in Java. The first one comes from the com.google.api.services.drive.model package whereas the second is from the java.io package. The last thing which is performed in this code snippet is presenting the obtained files list in the TextView named output. Since all the code is run in the separate thread, there is need to use the runOnUiThread method which modifies the UI using the main app thread. At this point there is one more thing to be discussed. The user of your app needs to give permission to use the Drive API on their behalf. Therefore the first time you invoke a Drive request using the execute() method, the UserRecoverableAuthIOException is thrown. You are obliged to catch this exception and start the new activity for the result, using the information provided by the exception object. The new activity asks the users permission and redirects control back to your app.

2.3 Uploading a file to the Drive

The second functionality of the Drive API is uploading files from the Android device to the specified

folder on the Drive. The code snippet responsible for this is provided below. As before, the whole code is invoked in the new thread.

```
Thread t = new Thread(new Runnable() {

            @Override
            public void run() {
                try {

                    FileList files = drive.files().list().setQ("title = 'MyFiles' and
trashed = false").execute();
                    String directoryId = "";

                    if(files.getItems().size()==0)
                    {
                        File appDir = new File();
                        appDir.setTitle("MyFiles");
                        appDir.setMimeType("application/vnd.google-apps.folder");
                        appDir.setDescription("Sample app folder");

                        File directory = drive.files().insert(appDir).execute();
                        directoryId = directory.getId();
                    } else {
                        directoryId = files.getItems().get(0).getId();
                    }

                    String randomUUID = UUID.randomUUID().toString();

                    final File sampleFile = new File();
                    sampleFile.setTitle("file"+randomUUID);
                    sampleFile.setMimeType("text/plain");

                    List<ParentReference> parents = new ArrayList<ParentReference>();
                    ParentReference fileParent = new ParentReference();
                    fileParent.setId(directoryId);
                    parents.add(fileParent);
                    sampleFile.setParents(parents);

                    File uploadedFile = drive.files().insert(sampleFile, new
ByteArrayContent("text/plain", randomUUID.getBytes())).execute();

                    if(uploadedFile!=null){
```

```
                        runOnUiThread(new Runnable() {

                            @Override
                            public void run() {
                                output.setText("Uploaded file " + sampleFile.getTitle() +
" to folder MyFiles");
                                Toast.makeText(MainActivity.this, "File successfully
uploaded", Toast.LENGTH_LONG).show();
                            }
                        });

                    }

                } catch (UserRecoverableAuthIOException e) {
                    startActivityForResult(e.getIntent(), REQUEST_AUTHORIZATION);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        });
        t.start();
```

[Code 4] Snippet of the MainActivity.java responsible for uploading new file to the Google Drive

The beginning of the presented snippet is similar to the beginning of the section looking at Listing files from a Drive. In this case we also use the Drive API to get a files list, but we invoke the additional method setQ on the Files object. This method allows specifying query parameters to get the filtered list of files and directories. There are set two parameters: title to MyFiles and trashed to false. Using these means we will receive the list of files and directories whose title contains the MyFiles string and which are not deleted. The next thing is checking if the directory named MyFiles exists. If it exists we simply get the id of this directory using the getId() method from the File class. The id is needed to place files in the directory. If there is no such directory, a new one is created and uploaded to the Drive. First we create the new File object, then we set its title, mime type and description. The mime type has to be set to the following value: application/vnd.google-apps.folder. The upload operation is performed using the insert(File f) method from the Files class and calling execute() on the Insert object. The execute method returns a newly created File object which contains the required id. The next step is to create the sample file and the relationship between this file and the MyFiles directory. The sample file is constructed in a way similar to the creating of the new directory. The main difference is the mime type. In the case of the sample file it is set to: text/plain. There are many other mime types which should be set appropriately to the file’s contents. When the file is ready, it is time to connect it with the MyFiles directory. It is achieved by creating a new ParentReference object with the id set to the directory’s id and adding this reference to the file’s parents list. The provided code snippet constructs a list of references, adds a newly created reference to it and invokes the setParents() method on the File object. When the file is ready and connected to its parent directory, it is time to upload it to the Drive. This is performed using the same insert() method which was used to create directory. The insert method is invoked with two

parameters: the first one is the File object to be uploaded, and the second is the content of this file. The one thing left to do is invoke the execute method which starts the upload process. When the network communication goes well, the uploaded file can be found in the MyFiles directory on the Drive. Additionally, the information about success is presented in the Android app.

3. Using the Google+ API

3.1 Getting started

The Google+ service also provides the API which can be used directly from the Android app. The library used in the sample app gives read-only access to the data from Google+. It is possible to share information from your app to Google+ under the condition that the native Google+ app is installed on the device. Such an approach is also presented in this subsection. The first steps are similar to using the Drive API. First of all, it is required to make sure that the user has choosen a Google account and we have the proper GoogleCredentialManager object. Additionally, the Google+ API libraries should be added to the Android project. The best way to do this, is by using the mentioned Google Plugin for Eclipse. You will also need to enable the Google+ API in the Google API Console. Having done that, you are ready to create the Plus instance which is needed to perform Google+ API communication. The Plus creation looks almost identical as the creating of a Drive instance and is therefore presented below without any further comments.

```
Plus plus = new Plus.Builder(AndroidHttp.newCompatibleTransport(), new GsonFactory(),
credential).build();
```

[Code 5] Creating Plus instance

3.2 Listing activities from the Google+

The provided sample app presents how to get the required information about the activities matching the android keyword from Google+. The code to perform this operation looks very similar to the one used to list files from the Drive. It is presented below and is followed by the explanation of the differences.

```
Thread t = new Thread(new Runnable() {

            @Override
            public void run() {
                try {
                    ActivityFeed feed = plus.activities().search("android").execute();
                    List<com.google.api.services.plus.model.Activity> items =
feed.getItems();

                    final StringBuilder sb = new StringBuilder();
                    sb.append("Activities connected to android:").append("\n\n");

                    for(com.google.api.services.plus.model.Activity activity : items) {

sb.append(activity.getTitle()).append("\n").append(activity.getUrl()).append("\n");
                    }
```

```
                    runOnUiThread(new Runnable() {

                        @Override
                        public void run() {
                            Toast.makeText(MainActivity.this, "Activities successfully
listed", Toast.LENGTH_LONG).show();
                            output.setText(sb.toString());
                        }
                    });


                } catch (UserRecoverableAuthIOException e) {
                    startActivityForResult(e.getIntent(), REQUEST_AUTHORIZATION);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        });
        t.start();
```

[Code 6] Snippet of the MainActivity.java responsible for listing activities from the Google+

The whole logic is the same as the logic used to list the files from Drive. The main difference is in the classes that we use. In this case we invoke the activities() method on the Plus instance which returns the Activities object. Then we invoke the search("android") method which tells the Activities object to search for the activities containing the android keyword. The last invoked method is execute() which sends the search request to the Google+ API and returns the ActivityFeed object. If you followed the part of the article concerning the Drive API the rest of the code should be straightforward for you. We retrieve items from the Activity Feed with the getItems() method and present them in the output TextView. Similarly to the Drive API, there may be a UserRecoverableAuthIOException thrown on the first invocation of the execute() method which should be handled as previously described.

3.3 Posting content to the Google+

The Google+ library, which is used in the sample app, provides read-only access to the data from Google+. It is possible to share the information from your Android app to Google+ with the help of the native Google+ app. To do this, you need to create a special intent which will trigger a part of the native Google+ app. Such an approach is presented below.

```
int errorCode = GooglePlusUtil.checkGooglePlusApp(this);

        if (errorCode != GooglePlusUtil.SUCCESS) {
          GooglePlusUtil.getErrorDialog(errorCode, this, 0).show();
        }
        else {
            Intent shareIntent =
```

```
ShareCompat.IntentBuilder.from(MainActivity.this).setType("text/plain").setText("This is
text posted from sample
application").getIntent().setPackage("com.google.android.apps.plus");

            startActivity(shareIntent);
        }
```

[Code 7] Snippet of the MainActivity.java responsible for firing Google+ share intent

To be able to fire the intent for sharing information with the native Google+ app you have to be sure that the app is installed. This check is performed using the checkGooglePlusApp(Context c) static helper method from the GooglePlusUtils class. This method returns code which is then compared to GooglePlusUtil.SUCESS code. If there is no Google+ app detected, the error dialog is presented. This is created with the getErrorDialog(…) helper method. In the case that the Google+ app is present, it is possible to fire the sharing intent. The Intent is build using a ShareCompat.IntentBuilder object. The most important part of it is the package name which has to be set with the setPackage(String s) method to the following value: com.google.android.apps.plus. Having the intent ready, you can start the Google+ app's activity for sharing info to Google+. This is done with the startActivity(Intent i) method. Such an approach is not ideal, but at least it allows your app's users to post information from your app to Google+.

Ref:http://developer.samsung.com/android/technical-docs/Using-Google-APIs