Authorization

Google Play services offers a standard authorization flow for all Google APIs and all components of Google Play services. In addition, you can leverage the authorization portion of the Google Play services SDK to gain authorization to services that are not yet supported in the Google Play services platform by using the access token to manually make API requests or using a client library provided by the service provider.

IN THIS DOCUMENT

For implementation details, see the sample in `<android-sdk>/extras/google-play-services/samples/auth`, which shows you how to carry out these basic steps for obtaining an acesss token.

Choosing an Account

Google Play services leverage existing accounts on an Android-powered device to gain authorization to the services that you want to use. To obtain an access token, a valid Google account is required and it must exist on the device. You can ask your users which account they want to use by enumerating the Google accounts on the device or using the built-in `AccountPicker` (/reference/com/google/android/gms/common/AccountPicker.html) class to display a standard account picker view. You'll need the `GET_ACCOUNTS` (/reference/android/Manifest.permission.html#GET_ACCOUNTS) permission set in your manifest file for both methods.

For example, here's how to gather all of the Google accounts on a device and return them in an array. When obtaining an access token, only the email address of the account is needed, so that is what the array stores:

```
private String[] getAccountNames() {
    mAccountManager = AccountManager.get(this);
    Account[] accounts = mAccountManager.getAccountsByType(
            GoogleAuthUtil.GOOGLE_ACCOUNT_TYPE);
    String[] names = new String[accounts.length];
    for (int i = 0; i < names.length; i++) {
        names[i] = accounts[i].name;
    }
    return names;
}
```

Obtaining an Access Token

With an email address and the service scope you can now obtain an access token.

> **Note:** Specify `"oauth2:`*`scope`*`"` for a single scope or `"oauth2:`*`scope1 scope2 scope3`*`"` for multiple scopes.

There are two general ways to get a token:

- Call one of the two overloaded `GoogleAuthUtil.getToken()` methods in a foreground activity where you can display a dialog to the user to interactively handle authorization errors.
- Call one of the three `getTokenWithNotification()` methods if you are trying to gain authorization in a background service or sync adapter so that a notification is displayed if an error occurs.

Using getToken()

The following code snippet obtains an access token with an email address, the scope that you want to use for the service, and a `Context`:

```
HelloActivity mActivity;
String mEmail;
String mScope;
String token;

...
try {
    token = GoogleAuthUtil.getToken(mActivity, mEmail, mScope);
} catch {
    ...
}
```

Call this method off of the main UI thread since it executes network transactions. An easy way to do this is in an `AsyncTask (/reference/android/os/AsyncTask.html)`. The sample in the Google Play services SDK shows you how to wrap this call in an AsyncTask. If authorization is successful, the token is returned. If not, the exceptions described in Handling Exceptions (#handle) are thrown that you can catch and handle appropriately.

Using getTokenWithNotification()

If you are obtaining access tokens in a background service or sync adapter, there are three overloaded `getTokenWithNotification()` (/reference/com/google/android/gms/auth/GoogleAuthUtil.html#getTokenWithNotification(android.content.Context, java.lang.String, java.lang.String, android.os.Bundle)) methods that you can use:

- `getTokenWithNotification(Context context, String accountName, String scope, Bundle extras)`: For background services. Displays a notification to the user when authorization errors occur.
- `getTokenWithNotification(Context context, String accountName, String scope, Bundle extras, Intent callback)`: This method is for use in background services. It displays a notification to the user when authorization errors occur. If a user clicks the notification and then authorizes the app to access the account, the intent is broadcasted. When using this method:
  - Create a `BroadcastReceiver` that registers the intent and handles it appropriately
  - In the app's manifest file, set the `android:exported` attribute to `true` for the broadcast receiver
  - Ensure that the intent is serializable using the `toUri(Intent.URI_INTENT_SCHEME)` and `parseUri(intentUri, Intent.URI_INTENT_SCHEME)` methods.
- `getTokenWithNotification(Context context, String accountName, String scope, Bundle extras, String authority, Bundle syncBundle)`: This method is for use in sync adapters. It displays a notification to the user when errors occur. If a user clicks the notification and then authorizes the app to access the account, the sync adapter retries syncing with the information contained in the `syncBundle` parameter.

  See the sample in `<android-sdk>/extras/google-play-services/samples/auth` for implementation details.

Handling Exceptions

When requesting an access token with `GoogleAuthUtil.getToken()` (/reference/com/google/android/gms/auth/GoogleAuthUtil.html#getToken(android.content.Context, java.lang.String, java.lang.String)), the following exceptions can be thrown:

- `UserRecoverableAuthException`: This exception is thrown when an error occurs that users can resolve, such as not yet granting access to their accounts or if they changed their password. This exception class contains a `getIntent()` method that you can call to obtain an intent that you can use with `startActivityForResult()` to obtain the user's resolution. You will need to handle the `onActivityResult()` callback when this activity returns to take action based on the user's actions.
- `GooglePlayServicesAvailabilityException`: This exception is a special case of `UserRecoverableAuthException` and occurs when the actual Google Play services APK is not installed or unavailable. This exception provides additional client support to handle and fix this issue by providing an error code that describes the exact cause of the problem. This exception also contains an intent that you can obtain and use to start an activity to resolve the issue.

- `GoogleAuthException`: This exception is thrown when the authorization fails, such as when an invalid scope is specified or if the email address used for authorization is actually not on the user's device.
- `UserRecoverableNotifiedException`: This exception is thrown when the authorization fails using one of the `getTokenWithNotification()` methods and if the error is recoverable with a user action.

For more information on how to handle these exceptions and code snippets, see the reference documentation for the `GoogleAuthUtil` (/reference/com/google/android/gms/auth/GoogleAuthUtil.html) class.

Using the Access Token

Once you have successfully obtained a token, you can use it to access Google services. Many Google services provide client libraries, so it is recommended that you use these when possible, but you can make raw HTTP requests as well with the token. The following example shows you how to do this and handle HTTP error and success responses accordingly:

```
URL url = new URL("https://www.googleapis.com/oauth2/v1/userinfo?access_token="
        + token);
HttpURLConnection con = (HttpURLConnection) url.openConnection();
int serverCode = con.getResponseCode();
//successful query
if (serverCode == 200) {
    InputStream is = con.getInputStream();
    String name = getFirstName(readResponse(is));
    mActivity.show("Hello " + name + "!");
    is.close();
    return;
//bad token, invalidate and get a new one
} else if (serverCode == 401) {
    GoogleAuthUtil.invalidateToken(mActivity, token);
    onError("Server auth error, please try again.", null);
    Log.e(TAG, "Server auth error: " + readResponse(con.getErrorStream()));
    return;
//unknown error, do something else
} else {
    Log.e("Server returned the following error code: " + serverCode, null);
    return;
}
```

Notice that you must manually invalidate the token if the response from the server signifies an authorization error (401). This could mean the access token being used is invalid for the service's scope or the token may have expired. If this is the case, obtain a new token using `GoogleAuthUtil.getToken()` (/reference/com/google/android/gms/auth/GoogleAuthUtil.html#getToken(android.content.Context, java.lang.String, java.lang.String)).