Implementing Player Sign-In in Android

It's important that users understand what the sign in process means and how it will make gameplay more enjoyable. These guidelines will help you ensure that your application provides a good sign in experience to the user.

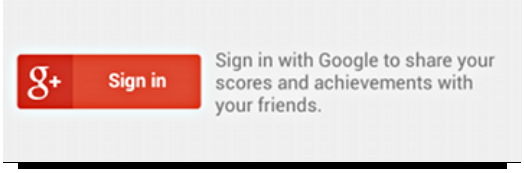
Provide a Sign-in button

Make sure that the sign-in button is visible to the user on your game's main screen or equivalent, and in all other screens where signing in might have an effect on gameplay. The button must be a standard **Google sign in** button, which can be rendered by either of the following methods:

- Include a `com.google.android.gms.common.SignInButton` on your layout as described in Initializing Your Games Client in Android; or
- Design your own sign-in button according to the Google+ branding guidelines.

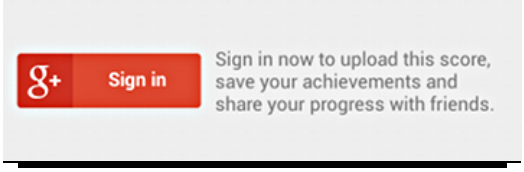
Clicking the **Sign in** button should initiate the sign in flow. If you are using the `BaseGameActivity` base class provided in the samples, simply call the `beginUserInitiatedSignIn()` method. Otherwise, you must manually call the `connect()` method of your `GamesClient` object.

In all cases, always provide some text next to the sign-in button explaining to the user what happens when they sign in. For example, consider how the sign-in button is presented in the "Type-a-Number" sample application:



Notice that the text next to the button clarifies why the user should click it, and how gameplay will be improved if they do.

When providing the sign-in button on other screens, use context to adjust the text so that it reflects what happens when the player signs in. Using "Type-a-Number" as an example again, this screenshot shows how to present the sign-in button on the screen that displays the user's score when the user has just won a game:



Keep the player signed in after the first time

After the user signed in for the first time in your application, you should always try to sign them in automatically when the app starts. Do this until the user explicitly signs out. If you are using the `BaseGameActivity` class from the samples, this is implemented for you automatically.

If you are not deriving your class from `BaseGameActivity`, you will need to implement this functionality in the `onStart()` method:

```
boolean mExplicitSignOut = false;
```

```
boolean mInSignInFlow = false; // set to true when you're in the middle of the
               // sign in flow, to know you should not attempt
               // to connect on onStart()
GamesClient mGamesClient;  // initialized in onCreate

@Override
protected void onStart() {
    super.onStart();
    if (!mInSignInFlow && !mExplicitSignOut) {
        // auto sign in
        mGamesClient.connect();
    }

@Override
public void onClick(View view) {
    if (view.getId() == R.id.sign_out_button) {
        // user explicitly signed out, so turn off auto sign in
        mExplicitSignOut = true;
        mGamesClient.signOut(this);
    }
    // ...
}
```

Make sign in optional

Users may not be comfortable signing in immediately after starting to play a new game. Some users may prefer to do so only after they have gained enough understanding and confidence about the game, while others may prefer never to sign in. We highly recommended that you implement player sign in as an optional (albeit encouraged) action.

Save player progress before sign in

If possible, save the user's progress locally while they are not signed in, and upload that progress when the user eventually signs in. That way, users will not lose any of their progress even if they decide to postpone signing in to your game. For example, both the "Type-a-Number" and "Collect All the Things" samples can save state locally so that progress is recorded even when `GamesClient` is disconnected. Once the `GamesClient` is connected, that progress is uploaded.

> **Note:** Although implementing sign in is optional, you should always make sure that the `GamesClient` is created and successfully connected to the game services.

You can check whether or not the user is signed in by calling the `isConnected()` method.

To reduce code complexity, you should create an object that stores the score and the achievements. This object can push those scores and achievements to a `GamesClient` when it's connected:

```
MyGameProgress mGameProgress = ....;

// when user finishes level:
mGameProgress.addScore(userScore);
mGameProgress.addAchievement(fooAchievement);
mGameProgress.addAchievement(barAchievement);
if (mGamesClient.isConnected()) {
    mGameProgress.save(mGamesClient);
}
```

The user's progress is stored in the `MyGameProgress` class. This allows you to store the progress for later upload, if the user is not signed in. When you detect that the user has signed in, you can call `mGameProgress.save(GamesClient)`:

```
@Override
protected void onSignInSucceeded() {  // from BaseGameActivity
    // sign in successful, so save progress if we have any.
    if (mGameProgress != null) {
        mGameProgress.save(mGamesClient)
    }
}
```

> **Note:** You should also serialize the `MyGameProgress` object to persistent local storage so that the user does not lose progress when your application terminates.

Remind users that they are signed in

During gameplay, you should present an appropriate reminder or hint to let players know if your game is performing some action on their behalf because they are signed in.

For example, when a signed-in user finishes a level, make it clear that the score and achievements are being automatically uploaded. As an example, here is the message that the "Type-a-Number" sample uses on the "win" screen:

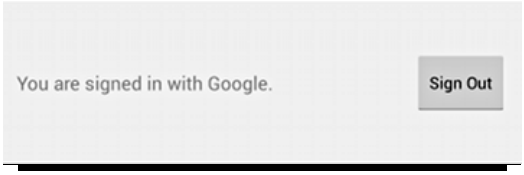
"You are signed in with Google. Your achievements and scores will be saved automatically."

If you are deriving your class from `BaseGameActivity`, you can simply query the value of the `mSignedIn` variable to determine whether the user is signed in or not. Otherwise, you can also check their sign in status by querying the `isConnected()` method of the `GamesClient` object.

```
if (mGamesClient.isConnected()) {
   // signed in. Show the "sign out" button and explanation.
   // ...
} else {
   // not signed in. Show the "sign in" button and explanation.
   // ...
}
```

Allow the user to sign out

After signing in, the user must always have the option to sign out. To do this, you can provide a "sign out" option or button on your main screen. For example, this is the sign out button used in the "Type-a-Number" sample:



To implement sign-out, simply call the `signOut()` method of `BaseGameActivity`.

If you are not using `BaseGameActivity` as a base class, call `GamesClient.signOut()` and remember to set a flag indicating that the user explicitly signed out, so that your `onStart` method knows not to automatically initiate the sign in process.

Last updated May 15, 2013.