I recently published a blog entry about lessons that I've learned during the course of creating Screebl , my entry in the Android Developer Challenge 2 . In my previous post, I mentioned that one of the top three things that I'd learned was "Polish is better than feature". My claim is that the majority of the applications that are available for Android right now lack polish, and consequently it is difficult to gain traction with users evaluating mobile applications.

NOTE: I've created a Google code project for this component, as well as a Google group for discussing issues. Please see the OpenYouTubeActivity projec t here on KeyesLabs.com.

In the interest of improving the Android platform, I'm going to be writing a series of articles that describe things that I think I've done well in Screebl , and I'm also going to contribute the code behind those features for others to use if they wish. I'm hoping that others in the development community will follow suit. It would be great if we had a market (as I mentioned here) specifically for delivering packaged widgets to other devs, but I'm not aware of anything at the moment, so I'll use this venue for now.

As I was putting the finishing touches on Screebl for ADC2, it occured to me that creating an amature video to describe the concepts and the basics functionality would be a good idea. Furthermore, it occured to me that it would be nice to offer the video to new users of the app the first time that they invoked the application on the mobile device.

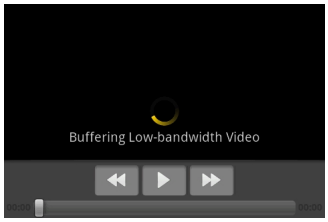
There is decent support in Android for media, including video, and it didn't take me long to catalog the options. None seemed to fit perfectly, however. Here were my requirements:

1. Stream the media. I didn't want to include the video as a resource for obvious reasons. Furthermore, I wanted the flexibility to be able to potentially make changes to the video after deployment of the application if I needed to.
2. Support low- and high-bandwidth scenarios. I didn't want the display of the video to require WiFi or 3G connectivity. I don't have 3G locally, and it's a huge pain to try and stream HD video.
3. Embedded workflow. I didn't want to lose control of the user when displaying a video. For example, registering an Intent that fired up the YouTube application left the user in the YouTube app to support their viral features. I wanted the user to return immediately after the

video completed to my application.

4. Support for video updates. If I discovered after deployment that my fly was unzipped in the video and wanted to update content, I needed to be able to upload a zipped-up version.
5. No backend server work. I didn't want to write a new server backend to do any of this.

YouTube seemed like the most obvious solution for the platform. How hard could it be? Well, the short answer is that it took me plenty of time, and fiddling, and playing, and hair pulling. I finally ended up with the **Activity** that you find here: **IntroVideoActivity** .



This class is meant to be included in your project, is easily invoked using a simple Intent, and can load a video from YouTube identified in one of the following two ways:

- Video Id. By including a URL in the Intent data of the form **ytv://videoid**, one can play a specific video.
- Playlist Id. By including a URL in the Intent data of the form **ytpl://playlistid**, one can play the latest video added to a YouTube playlist. This is great for adding new video content that you want your users to see later on. Simply add a new video to the YouTube playlist, and the next time that the user invokes the activity pointing at that playlist, they will see the new video.

The Activity takes care of doing all of the tricky YouTube token negotiation. It also uses a simple algorithm for detecting bandwidth available at the client, and adjusts the quality of the downloaded video appropriately.

Invoking the activity is easy. Simply use a code snippet similar to the following:

```
Intent lVideoIntent = new Intent(null, Uri.parse("ytpl://"+YOUTUBE_PLAYLIST_ID), this, IntroVideoActivity.class);
startActivity(lVideoIntent);
```

There is other coniguration that you can do with respect to the messages that get displayed to

the user, but those are well documented in the Javadoc.

You can download the source code for **IntroVideoActivity.java** here . This code has a liberal license, so use it as you will if you find it matches your use cases. My only requests:

- Don't expect lots of support from me.
- Reciprocate with your own clever widgets, tricks, and genius insights.
- Download my apps and review them well.