Week 15

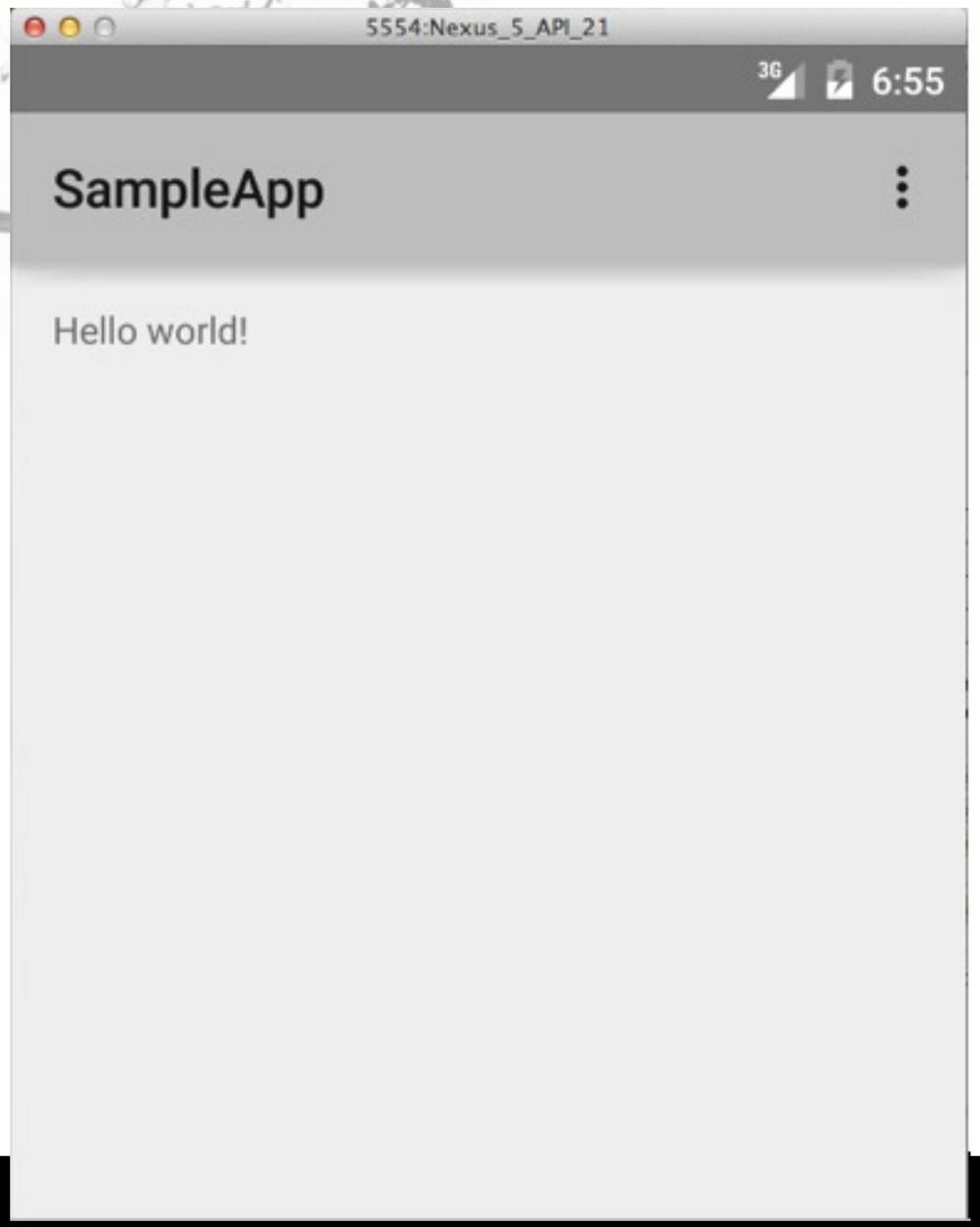
@wesreisz

Agenda

- Broadcast Receiver
- Storage Options
- Saving Data
- Mobile Backend as a Service

Installing and Updating to Lollipop

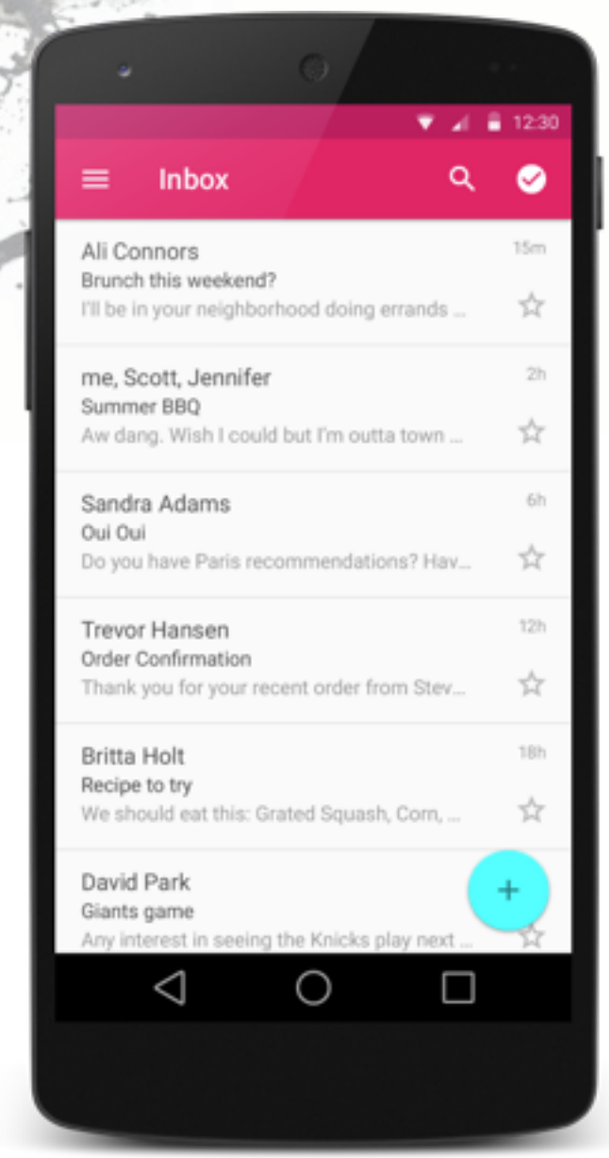
- New Wizards
- New Material Styles (Dark and Light)
- Intel HAXM
- *-v21

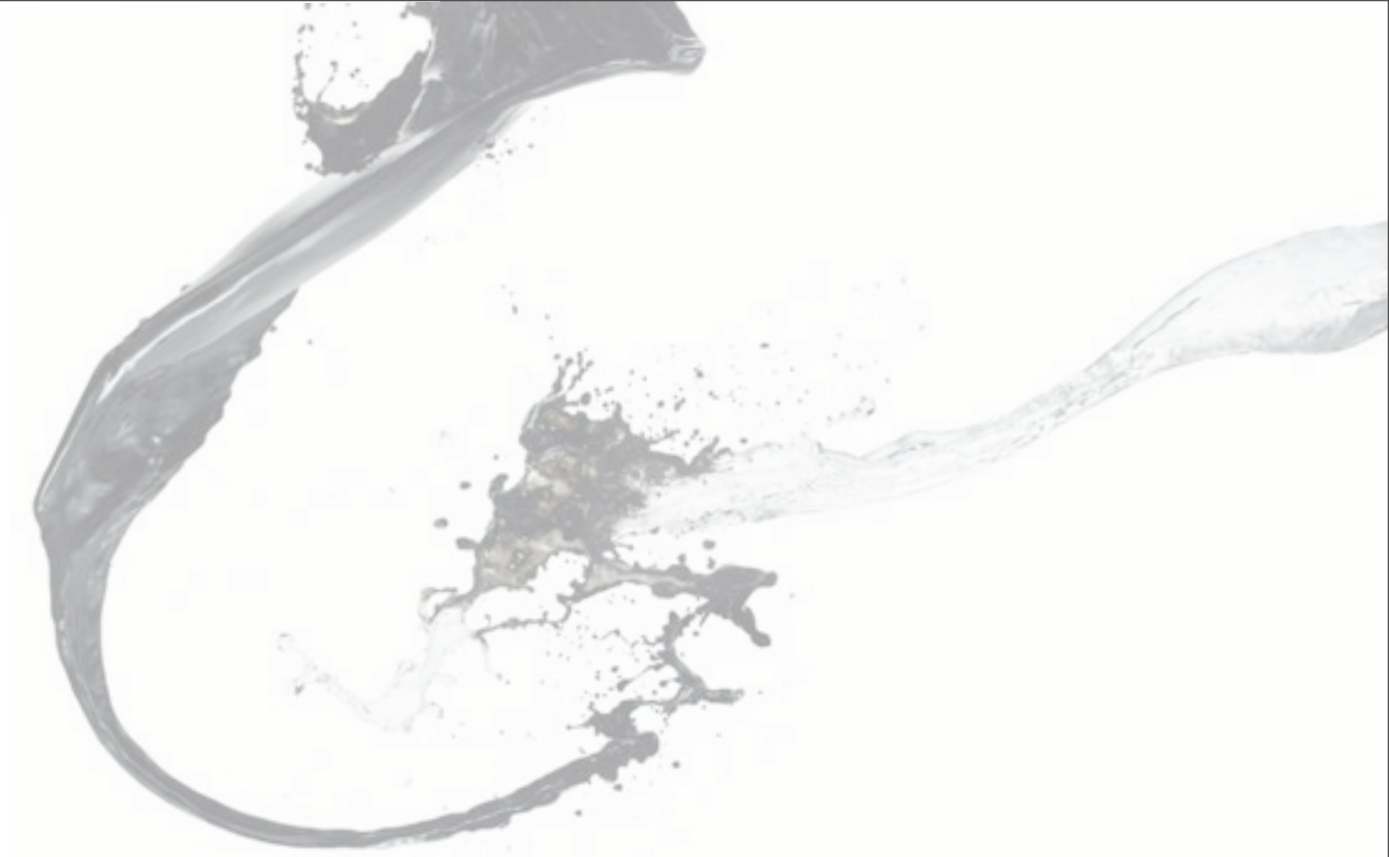


WESLEY REISZ

Material Design

- https://developer.android.com/design/material/index.html





Broadcast Receiver

Broadcast Receiver

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >

   <receiver android:name="MyReceiver">
      <intent-filter>
         <action android:name="android.intent.action.BOOT_COMPLETED">
      </action>
      </intent-filter>
   </receiver>

</application>
```

```
public class MyReceiver extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
   }

}
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
   <application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name=".MainActivity"
           android:label="@string/title_activity_main" >
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER"/>
           </intent-filter>
       </activity>
       <receiver android:name="MyReceiver">
          <intent-filter>
          <action android:name="com.tutorialspoint.CUSTOM
          </action>
          </intent-filter>
      </receiver>
   </application>
</manifest>
```

```
package com.example.helloworld;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class MyReceiver extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
   }

}
```

```
public void broadcastIntent(View view)
{
   Intent intent = new Intent();
   intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
   sendBroadcast(intent);
}
```

Storage Options

Storage Options

- **Shared Preferences**
- Internal Storage
- External Storage
- SQLite Database
- Network Connection

Shared Preferences

The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of **primitive data types**.

You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).

SharedPreferences

- Couple ways to get access to shared preferences:

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

"com.example.myapp.PREFERENCE_FILE_KEY"

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
```

Shared Preferences

- Reading Shared Preferences

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue = getResources().getInteger(R.string.saved_high_score_default);
long highScore = sharedPref.getInt(getString(R.string.saved_high_score), defaultValue);
```

Shared Preferences

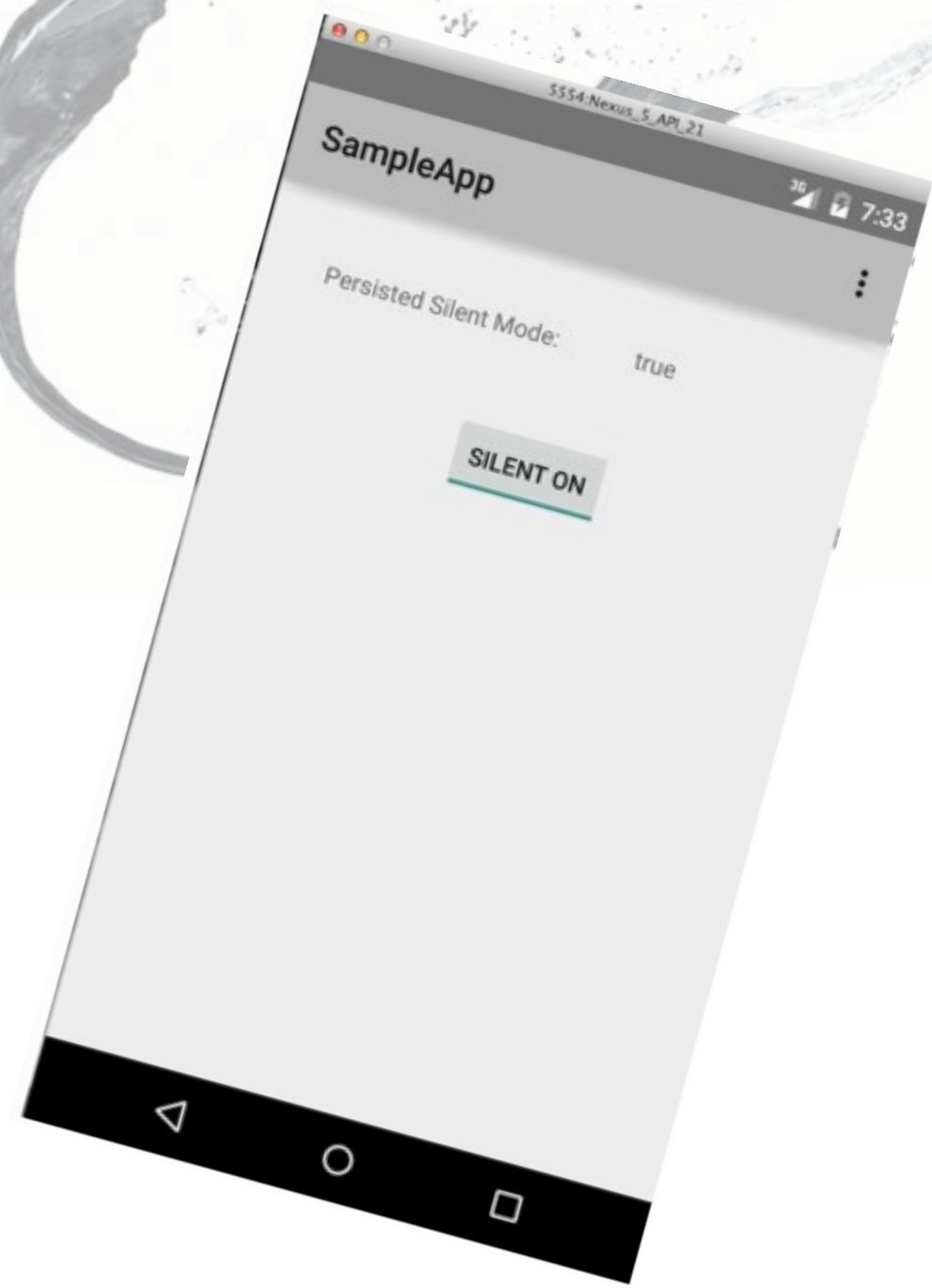
- Writing to Shared Preferences

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

Full Example

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
        super.onCreate(state);
        . . .

        // Restore preferences
        SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
        boolean silent = settings.getBoolean("silentMode", false);
        setSilent(silent);
    }

    @Override
    protected void onStop(){
        super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

Demo



WESLEY REISZ

Storage Options

- Shared Preferences
- **Internal Storage**
- External Storage
- SQLite Database
- Network Connection

Internal Storage

You can save files directly on the device's internal storage.

By default, files saved to the internal storage are **private to your application** and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.

Steps

- To create and write a private file to the internal storage:
  - Call openFileOutput() with the name of the file and the operating mode. This returns aFileOutputStream.
  - Write to the file with write().
  - Close the stream with close().

Writing

```
public void saveSongs(List<Song> songs) throws  JSONException, IOException{
    ObjectWriter ow = new ObjectMapper().writer();
    String json = ow.writeValueAsString(songs);

    Writer writer = null;
    try{
        OutputStream out = mContext
            .openFileOutput(mFileName,Context.MODE_PRIVATE);
        writer = new OutputStreamWriter(out);
        writer.write(json);

    }finally{
        if(writer!=null){
            writer.close();
        }
    }
}
```

https://github.com/introandroidonline/MyMusicList/blob/master/mymusiclist/src/main/java/com/learninghouse/mymusiclist/SongJSONSerializer.java

Writing

```
public void saveSongs(List<Song> songs) throws  JSONException, IOException{
    ObjectWriter ow = new ObjectMapper().writer();
    String json = ow.writeValueAsString(songs);

    Writer writer = null;
    try{
        OutputStream out = mContext
            .openFileOutput(mFileName,Context.MODE_PRIVATE);
        writer = new OutputStreamWriter(out);
        writer.write(json);

    }finally{
        if(writer!=null){
            writer.close();
        }
    }
}
```

btw, this is an example of Jackson, mapping an object into a json string.

https://github.com/introandroidonline/MyMusicList/blob/master/mymusiclist/src/main/java/com/learninghouse/mymusiclist/SongJSONSerializer.java

Reading

```
public List<Song> loadSongs() throws IOException, JSONException{
    List<Song> songs = new ArrayList<Song>();
    BufferedReader reader = null;
    try{
        InputStream in = mContext.openFileInput(mFileName);
        reader = new BufferedReader(new InputStreamReader(in));
        String json = null;
        if((json=reader.readLine())!=null){
            ObjectMapper mapper = new ObjectMapper();
            songs = mapper.readValue(json, mapper.getTypeFactory().constructCollectionType(List.class, Song.class));
        }
    }catch(FileNotFoundException e){
        //ignore happens on first load
    }finally{
        if(reader!=null){
            reader.close();
        }
    }

    return songs;
}
```

https://github.com/introandroidonline/MyMusicList/blob/master/mymusiclist/src/main/java/com/learninghouse/mymusiclist/SongJSONSerializer.java

Reading

```
public List<Song> loadSongs() throws IOException, JSONException{
    List<Song> songs = new ArrayList<Song>();
    BufferedReader reader = null;
    try{
        InputStream in = mContext.openFileInput(mFileName);
        reader = new BufferedReader(new InputStreamReader(in));
        String json = null;
        if((json=reader.readLine())!=null){
            ObjectMapper mapper = new ObjectMapper();
            songs = mapper.readValue(json, mapper.getTypeFactory().constructCollectionType(List.class, Song.class));
        }
    }catch(FileNotFoundException e){
        //ignore happens on first load
    }finally{
        if(reader!=null){
            reader.close();
        }
    }

    return songs;
}
```

This is an example of mapping a string to a list of a custom class using jackson

https://github.com/introandroidonline/MyMusicList/blob/master/mymusiclist/src/main/java/com/learninghouse/mymusiclist/SongJSONSerializer.java

Jackson Object Mapping

- Demo
  - Tutorial:
    http://wiki.fasterxml.com/JacksonInFiveMinutes
  - Generate POJO(s):
    http://www.jsonschema2pojo.org
    http://jsongen.byingtondesign.com/

Jackson Object Mapping

Have JSON

```
{
  "name" : { "first" : "Joe", "last" : "Sixpack" },
  "gender" : "MALE",
  "verified" : false,
  "userImage" : "Rm9vYmFyIQ=="
}
```

It takes two lines of Java to turn it into a User instance:

Create Object

Toggle line numbers

```
1 ObjectMapper mapper = new ObjectMapper(); // can reuse, share globally
2 User user = mapper.readValue(new File("user.json"), User.class);
```

Map in Code

Storage Options

- Shared Preferences
- Internal Storage
- **External Storage**
- SQLite Database
- Network Connection

External Storage

Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

Caution: External storage can become unavailable if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files you save to the external storage. All applications can read and write files placed on the external storage and the user can remove them.

WESLEY REISZ

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
```

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

Storage Options

- Shared Preferences
- Internal Storage
- External Storage
- SQLite Database
- Network Connection

SQLite

- What is SQLite?
  - SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private.

SQLite

- What is the main package for SQLite in Android?
  - The main package is android.database.sqlite that contains the classes to manage your own databases
  - http://developer.android.com/reference/android/database/sqlite/package-summary.html

SQLite

- Focus on the details next week with content providers

Storage Options

- Shared Preferences
- Internal Storage
- External Storage
- SQLite Database
- **Network Connection**

Network Connection

- AsyncTask or Thread
- and HttpClient with Get/PUT/POST/Delete

```java
public static String getJson(String url){
    InputStream inputStream = null;
    String result = "";
    try {
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse httpResponse = httpclient.execute(new HttpGet(url));
        inputStream = httpResponse.getEntity().getContent();

        // convert inputstream to string
        if(inputStream != null)
            result = convertInputStreamToString(inputStream);
        else
            result = "Did not work!";

    } catch (Exception e) {
        Log.d("InputStream", e.getLocalizedMessage());
    }

    return result;
}
```

Backend as a Service (Baas)

Backend as a Service

Mobile Backend as a service(MBaaS), also known as " backend as a service" (BaaS), is a model for providing web and mobile app developers with a way to link their applications to backend cloud storage and APIs exposed by back end applications while also providing features such as user management, push notifications, and integration with social networking services.

These services are provided via the use of custom software development kits (SDKs) and application programming interfaces (APIs). BaaS is a relatively recent development in cloud computing,[5] with most BaaS startups dating from 2011 or later.

Although a fairly nascent industry, trends indicate that these services are gaining mainstream traction with enterprise consumers. The global BaaS market had an estimated value of $216.5 million in 2012 and projected to grow to $7.7 billion by 2017.

API Docs

Parse

Dashboard Quickstart Tutorials Documentation Downloads Help Pricing Products

Welcome Wes Reisz

Search Docs

- Overview
- Push Notifications
- iOS/OS X
- Android
  - Objects
    - The ParseObject
    - Saving Objects
    - Retrieving Objects
    - Saving Objects Offline
    - Updating Objects
    - Deleting Objects
    - Relational Data
    - Data Types
    - Other Operations
  - Queries
  - Subclasses
  - Files

Storing data on Parse is built around the `ParseObject`. Each `ParseObject` contains key-value pairs of JSON-compatible data. This data is schemaless, which means that you don't need to specify ahead of time what keys exist on each `ParseObject`. You simply set whatever key-value pairs you want, and our backend will store it.

For example, let's say you're tracking high scores for a game. A single `ParseObject` could contain:

```
score: 1337, playerName: "Sean Plott", cheatMode: false
```

Keys must be alphanumeric strings. Values can be strings, numbers, booleans, or even arrays and objects - anything that can be JSON-encoded.

Each `ParseObject` has a class name that you can use to distinguish different sorts of data. For example, we could call the high score object a `GameScore`. We recommend that you NameYourClassesLikeThis and nameYourKeysLikeThis, just to keep your code looking pretty.

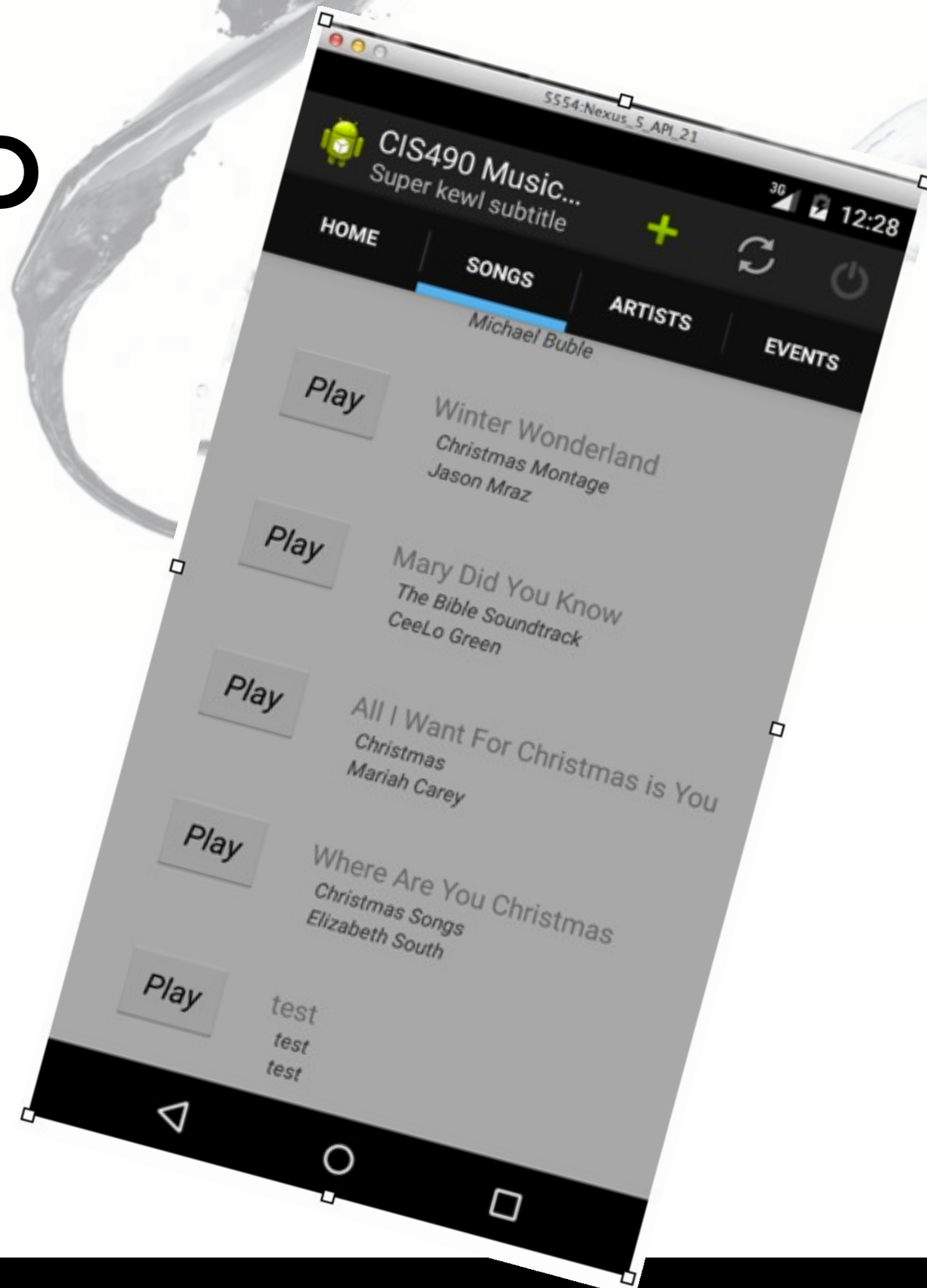
Saving Objects

Let's say you want to save the `GameScore` described above to the Parse Cloud. The interface is similar to a `Map`, plus the `saveInBackground` method:

```
ParseObject gameScore = new ParseObject("GameScore");
gameScore.put("score", 1337);
gameScore.put("playerName", "Sean Plott");
gameScore.put("cheatMode", false);
```

https://parse.com/docs/android_guide

Demo



WESLEY REISZ

Agenda

- Storage Options
- Saving Data
- Broadcast Receiver
- Mobile Backend as a Service