Parsing JSON

Web services are interfaces that are provided by web applications so that other programs may access data stored in each application. For example, the Google Maps API [1] is a web service that provides access to the data in Google Maps.

Web services typically behave similarly to normal web pages. Data is first retrieved from a URL (e.g. <`http://maps.googleapis.com/maps/api/`*`service`*`/`*`output`*`?`*`parameters`*>). This data is usually provided in a text-based format, commonly either XML (eXtensible Markup Language) [2] or JSON (JavaScript Object Notation) [3], which may be "parsed" so that the information in the format may be used by the application. In this lab, we will be working with data in the JSON format, which is more commonly used than XML [4].

JSON is a compact format which may be used to represent data in the basic types of *strings*, *numbers*, and *Boolean (true/false) values*, as well as a *null* value. JSON also supports two compound types: *objects (maps)* and *arrays*. JSON is based on JavaScript (C-like) syntax, so the basic *string*, *number*, and *boolean* values look much like they do in C:

- `"string"` is a *string* containing the characters "s", "t", "r", "i", "n", and "g" in that order. C-like character escapes are also supported (e.g. `"\n"` represents a line-break, while `"\\"` represents a single backslash character and `"\""` represents a literal double-quote). Unlike C, however, the `"\x"` escape is not supported for hexadecimal data, so binary data is not supported in JSON.
    - Strings *do* support Unicode characters both as literal characters and using the escape `"\u"` followed by four hexadecimal digits representing the Unicode character code (e.g. both `"अ"` (in the UTF-8 encoding) and `"\u0905"` represent the Devanagari character "अ")
- `10` is a *number* representing the integer 10, while `-3.14` is a *number* representing the decimal number -3.14. Exponents in base-10 may be constructed using the letter `e` (e.g. `1.6e-6` for 1.6×10^{-6})
- `true` is the *Boolean true value*, while `false` is the *Boolean false value*.
- `null` is the *null value*.

The compound types of *objects* and *arrays* more closely resemble JavaScript, but still may look familiar to Python or PHP developers:

- *Arrays* begin and end with square brackets, and items in the array are separated by commas. For example, `["string", 1, true]` is an array containing three items: the *string* "string", the *number* 1, and the *Boolean true value* in that order.

---

[1] <https://developers.google.com/maps/documentation/webservices/>

[2] <http://www.w3schools.com/xml/xml_whatis.asp> is a nice overview of XML.

[3] <http://www.json.org/> is the home of the JSON standard.

[4] <http://www.jondev.net/articles/Android_XML_SAX_Parser_Example> provides a short example of using XML with Android, while IBM has a longer, more detailed, example on their developerWorks website at <http://www.ibm.com/developerworks/opensource/library/x-android/>.

- *Objects* are like C++ maps or Python dictionaries and contain key-value pairs. Keys are always strings. JSON objects begin and end with curly brackets. A single key-value pair has its key first, followed by a colon and its value (e.g. `{"key": "value"}` is an *object* containing a single key-value pair whose key is the *string* "key" and whose value is the *string* "value"). Multiple key-value pairs are separated by commas, just like items in an array (e.g. `{"a": 1, "b": 2}`).

On Android, JSON support is provided by a version of the `org.json` package [5]. This package is very easy to use once you have obtained the JSON data as a `String` object, provided that you know what data type will be returned.

Basic types such as *strings*, *numbers*, and *Boolean values*, are represented in Java as their corresponding types (`String`, `int`/`double`, `boolean`, respectively). Compound types are represented using classes in the `org.json` package. JSON arrays are represented by the class `org.json.JSONArray`; JSON objects are represented by the class `org.json.JSONObject`. *Null values* are represented by the instance `JSONObject.NULL`.

To parse compound JSON data from a `String`, you may simply create a new Java object of the appropriate type, passing the `String` as the only argument to the constructor.

For example, if a JSON *object* is in the `String jsonString`, then `org.json.JSONObject` may be used to parse it using the following code:

```
import org.json.JSONObject;

JSONObject jsonData = new JSONObject(jsonString);
```

Likewise, if a JSON *array* is in the `String jsonString`, then `org.json.JSONArray` may be used to parse it using the following code:

```
import org.json.JSONArray;

JSONArray jsonData = new JSONArray(jsonString);
```

You will generally never come across a base type represented in JSON from a web service. Every JSON web service will return either a JSON *object* or a JSON *array*.

Retrieving data from a `JSONObject` or `JSONArray` is also easy.

- Values for the keys in a `JSONObject` may be obtained using `get*(String key)` methods (e.g. `getBoolean(key)` will get a `boolean` value, `getInt(key)` will get an `int` value, `getString(key)` will get a `String` value, `getJSONObject(key)` will get a `JSONObject` value, and so on).
    - The basic `get(String key)` method will return an appropriate Java `Object` value (e.g.

[5] <http://www.json.org/java/index.html>, but also see <http://developer.android.com/reference/org/json/package-summary.html> in the Android documentation.

`Boolean`, `Integer`, `String`, `JSONObject`, `JSONObject.NULL`, etc.) if you do not know what type to expect. You can then use the **`instanceof`** operator to test the class of the object returned.

- `isNull(String key)` may be used to test if the value of a key is null. It will also return `true` if `key` does not exist in the `JSONObject`.
- *NOTE:* The `get*(String key)` methods will throw a `JSONException` if `key` is not found or if the value is not of the right type! Use `has(String key)` to test if a key exists, or the related `opt*(String key)` methods which will return a default value if the key does not exist or the value is of the wrong type.

- `keys()` will return an iterator Java object (`java.util.Iterator`) which you can use to iterate through the keys in a `JSONObject`.
- Values in a `JSONArray` may be obtained using `get*(int index)`, `isNull(int index)` and `opt*(int index)` which behave similarly to the `JSONObject` versions, except that they take an integer index into the *array*.
- Both `JSONObject` and `JSONArray` provide the `count()` method to return the number of items in the *object*/*array*.

As an example, here is some code which tries to get data from our `JSONObject jsonData`.

```
import org.json.JSONObject;

String stringValue;
int intValue;
boolean optionalWasNull = false;
try {
    // Set stringValue to the value for the key "string" or throw an
    // exception if it is not present, or is not a string.
    stringValue = jsonData.getString("string");

    // Set intValue to the value for the key "int" or the value 3 if
    // it is not present, or is not an integer.
    intValue = jsonData.optInt("int", 3);

    if (jsonData.has("optional") && jsonData.isNull("optional")) {
        // If the key "optional" is present and has a null value
        // (isNull() returns true if "optional" is not present)
        // we will execute this code.
        optionalWasNull = true;
    }
} catch (JSONException e) {
    // We will jump here if the key "string" is not present or does
    // not have a string value.
}
```

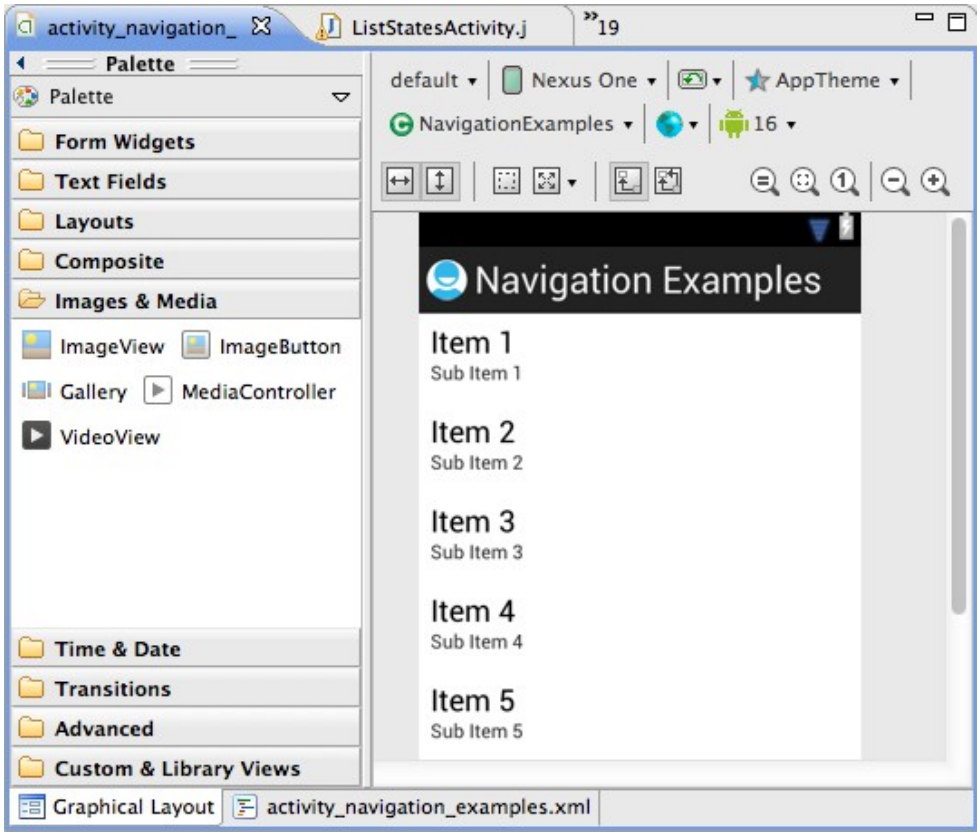

Illustration 1: The Graphical Layout window, with `ImageView` visible

Working with ImageViews and Bitmaps

`ImageView` views are used to display static images in Android. They are available under the *Images & Media* tab under the graphical layout editor. `ImageViews` may have their image content set at design time as a property, `src`, or in the code using the `setImageDrawable(Drawable d)` or `setImageBitmap(Bitmap b)` methods of the `ImageView`.

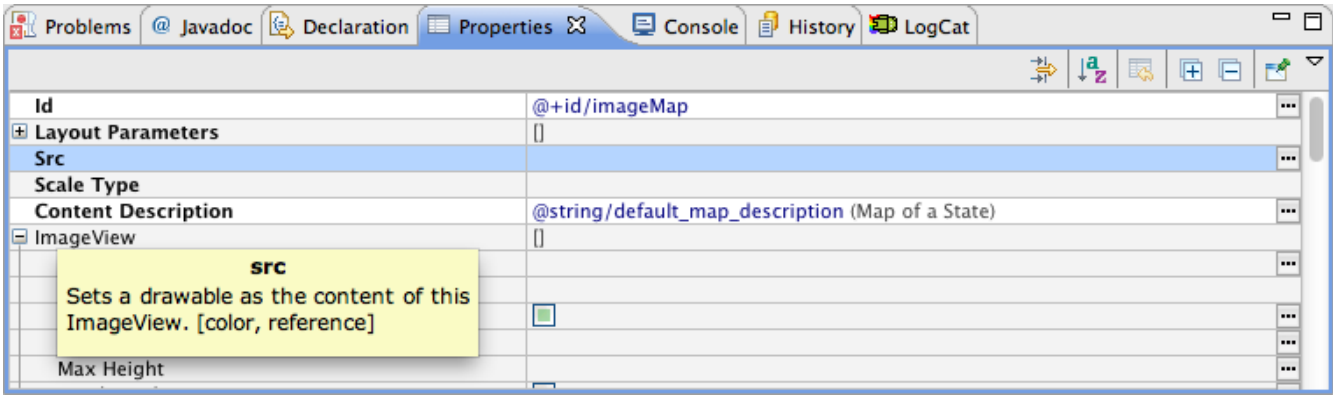

Illustration 2: Setting the `src` property from the Properties tab

The `src` property or `setImageDrawable()` method may be used to set the image to the contents of a *drawable resource* [6]. These drawable resources (e.g. PNG, JPEG, and GIF files) are stored in the

[6] <http://developer.android.com/guide/topics/resources/drawable-resource.html> has more information about drawable resources.

drawable-* directories in your res/ directory [7], and are mapped to resource ids based on their file name. For example, a file named res/drawable/*image*.jpg will be named @drawable/*image* when used in XML files (such as in the Graphical Layout editor). In your code, such a drawable may be referenced as R.drawable.*image* in your code. (*NOTE:* Just as string resources may be retrieved as String objects by using [getResources().]getString(R.string.myString), so may drawable resources be retrieved as Drawable objects using [getResources().]getDrawable(R.drawable.image).)

Any images that are *not* a part of your project (e.g. images you retrieve from the Internet) may be used in the form of Bitmap objects. Bitmap objects are usually constructed using the BitmapFactory class. Once you have obtained your data as an array of bytes such as by writing incoming binary data from an InputStream in to a ByteArrayOutputStream out:

```
byte[] byteBuffer = new byte[256];
while (true) {
    int bytesRead = in.read(byteBuffer);  // Read into byteBuffer
    if (bytesRead < 0) break;
    out.write(byteBuffer, 0, bytesRead);  // Write only bytes read
}
byte[] data = out.toByteArray();
```

it is possible to construct a Bitmap object using the BitmapFactory.decodeByteArray() method:

```
BitmapFactory.Options options = new BitmapFactory.Options();
Bitmap bitmap = BitmapFactory.decodeByteArray(
    data, 0, data.length, options);
```

The Bitmap returned may then be passed as the argument to the ImageView's setImageBitmap() method.

[7] The different drawable-* directories (hdpi, ldpi, mdpi, xhdpi) relate to the resolution of the screen. For more information about supporting different screen resolutions, check out <http://developer.android.com/guide/practices/screens_support.html>.