JSON and Django

In Lab 2, we discussed how to parse data in the JSON format on Android, but we said nothing about how to create new JSON data. Now that we have the tools that allow us to build web applications, we can also think about building our own web services on top of Django that use data in the JSON format both as a way to expose data for consumption by Android applications, but also as a way to receive data *from* Android apps. We'll consider each of these tasks in turn.

Sending JSON from Django

Django comes packaged with its own JSON libraries that you can use to parse and construct your own JSON data. This module, available as `django.utils.simplejson`, works with native Python types, translating them to and from the JSON format. Creating JSON data as a string may be done with the `dumps(obj)` method in the `simplejson` module. It is probably easiest to construct JSON data by passing Python lists and dictionaries to his method. For example,

```
import django.utils.simplejson as json

json_data = json.dumps(["string", 1, 2.5, None])
# json_data is now the string "[\"string\", 1, 2.5, null]"

json_data = json.dumps({"key1": 1, "key2": 2})
# json_data is now the string "{\"key1\": 1, \"key2\": 2}"
```

The string data in `json_data` may then be passed as the data argument to the `HttpResponse()` constructor in a view.

The `HttpResponse` class normally adds an *HTTP header* which describes the type of the data returned. Since most views return HTML data, `HttpResponse` automatically marks the data as HTML data. In order to better support JSON, however, it is generally good practice to change the "content-type" of the response so that the end-user does not see it as HTML data. To do so, you should pass an additional, named, argument to the `HttpResponse()` constructor: `content_type`. (e.g. `HttpResponse(json_data, content_type="text/html")`)

The argument `content_type` takes a string containing a valid MIME type (e.g. `"text/html"` for HTML, `"image/png"` for a PNG image, and so on). The best MIME type for JSON data is `"application/json"` as it is the official MIME type for JSON data. Unfortunately, if you wish to test the view in your browser, this MIME type will often download the data rather than displaying it. If you wish to display it, however, you may instead want to use the `"text/plain"` MIME type which represents unformatted raw text.

Making JSON Data in Android

There are several ways to construct JSON data in Android, depending on what data you are trying to construct. For example, `org.json.JSONObject` and `org.json.JSONArray` both have constructors which take the equivalent Java data structures (`java.util.Map` and `java.util.Collection` respectively) and construct the JSON Java objects. These may then be converted to a JSON `String` using the `toString()` method of the JSON Java object. For example:

```
import java.util.ArrayList;
import java.util.HashMap;
import org.json.JSONArray;
import org.json.JSONObject;

ArrayList<Integer> collection = new ArrayList<Integer>();
collection.add(Integer.valueOf(1));
collection.add(Integer.valueOf(2));

JSONArray jsonArray = new JSONArray(collection);
String jsonString = jsonArray.toString();
# jsonString is the string "[1, 2]"

HashMap<String, Object> map = new ArrayList<String, Object>();
map.put("key1", "string");
map.put("key2", Double.valueOf(2.5));

JSONObject jsonObject = new JSONObject(map);
jsonString = jsonObject.toString();
# jsonString is the string "{\"key1\": \"string\", \"key2\": 2.5}"
```

Another class, `org.json.JSONStringer`, may be used to construct JSON data piece-by-piece rather than by constructing an intermediate `JSONArray` or `JSONObject`. Objects of this class have 6 basic methods from which JSON may be constructed in-order. Four of these methods are used to mark the start and end of compound data structures in JSON (and should thus be the first and last methods called on the `JSONStringer` object before turning it into a string by `toString()`):

- `array()` – marks the start of a JSON array.
- `endArray()` – marks the end of a JSON array previously started with `array()`.
- `object()` – marks the start of a JSON object.
- `endObject()` – marks the end of a JSON object previously started with `object()`.

Two other methods are used to populate these compound data structures:

- `key(String string)` – Add a new key to a JSON object previously started with `object()`. (*NOTE:* the previous method should NOT have been `array()` or `key()`)
- `value(boolean/double/long/String v)` – Add a new value to a JSON object (in which case `key()` should have been the previous method) or JSON array.

`array()` and `object()` may also be used to mark the beginning of nested array or object values (instead of `value()`).

All of these methods return the same JSONWriter instance, so they may be chained.

We may thus construct the same two values of jsonString as follows:

```
import org.json.JSONStringer;

JSONStringer s = new JSONStringer();
s.array().value(1).value(2).endArray();
String jsonString = s.toString();
# jsonString is the string "[1, 2]"

s = new JSONStringer();
s.object().key("key1").value("string")
          .key("key2").value(2.5).endObject();
jsonString = s.toString();
# jsonString is the string "{"key1":"string","key2":2.5}"
```

This string may then be sent to your Django web service.

Sending data over an `HttpURLConnection`

Previously, in lecture, we discussed reading data from an `HttpURLConnection`. In this section, we will cover the opposite activity, uploading data (which may be serialized in JSON format) through an `HttpURLConnection`.

Sending data over an `HttpURLConnection` is similar in many ways to getting data from it; instead of getting an `InputStream` to read from the `HttpURLConnection`, we get an `OutputStream` to write to the `HttpURLConnection`.

The `HttpURLConnection` is opened in much the same way as when we wish to read data. We specify the `URL` to which we are going to upload our data (e.g. the Django view which reads the `request.POST` property) and connect to it using `openConnection()` which returns the `HttpURLConnection`. We also need to mark the connection as one on which we will be sending data. To do this, we must call `setDoOutput(true)` and we must set the size of the data we will be uploading (in bytes) with `setFixedLengthStreamingMode(bytes.length)` where we obtain `bytes` with `string.getBytes()`. Instead of getting the `InputStream` with `getInputStream()` however, we get the `OutputStream` with `getOutputStream()`. Finally, to send data to the server, we call the `write()` method of the `OutputStream` with the `byte[]` we wish to send before disconnecting the `HttpURLConnection`.

```
HttpURLConnection c;
OutputStream os;
byte[] data = jsonData.getBytes();
try {
    URL url = new URL("http://www.example.com/example_upload_site");
    c = url.openConnection();
    c.setDoOutput(true);
    c.setFixedLengthStreamingMode(data.length);
    os = c.getOutputStream();
} catch (MalformedURLException e) {
    // If the URL is not a valid URL.
} catch (IOException e) {
    // If an error prevented opening or writing to the stream.
}
try {
    os.write(data);
} catch (IOException e) {
    // If an error prevented opening or writing to the stream.
} finally {
    // This clause ensures that disconnect() is always run last,
    // even after an exception is caught.  You should wrap
    // reader.readLine() like this too.
    c.disconnect();
}
```

Note that this code will upload *exactly* the contents of the `jsonData` variable, and thus it is not suitable for submitting form data (such as to an HTML form). If we wish to read the response data from the server after sending our data, we need only get the `InputStream` of the connection after we have written all of the data, and read from that stream prior to disconnecting.

Receiving and Parsing JSON data in a Django View

We have already seen how a Django view reads input data into a `Form` object by testing `request.method` (to see if it equals `"POST"`) and then retrieving the data from the `request.POST` property. We cannot use the latter method to get the raw JSON data that might be uploaded by our Android application, but we can instead "read" the request by calling `request.read()` and use the returned string value to parse JSON uploaded to Django. We can parse the JSON input by then using another method of the `django.utils.simplejson` module, `loads()`, which takes, as its argument, the string of JSON data, and returns the data structure stored in the string (either a `dict` or `list`)

```
import django.utils.simplejson as json

json_data = request.read()
# json_data contains the data uploaded in request
data = json.loads(json_data)
# data is now a Python list or dict representing the uploaded JSON.

data = json.loads("[\"string\", 1, 2.5, null]")
# data is now the list ["string", 1, 2.5, None]

data = json.loads("{\"key1\": 1, \"key2\": 2}")
# data is now the dict {"key1": 1, "key2": 2}
```