XML and JSON

Slides adapted from content by Victor Matos.

Why XML and JSON?

- Standardized formatting for data interchange.
- Simple and human-readable.
- Other serialization mechanisms exist (protocol buffers) but are less popular.

XML

- Extensible Markup Language (XML)
- Established by the W3C organization
- Provide a framework for uniformly encoding documents in a human readable form
- Widely used
- Android OS relies heavily on XML to save its various resources such as layouts, manifest, etc.

Example XML document

```
<activity android:name="com.myapp.MainActivity"
          android:label="MyApp" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Key terms:

Element: <activity>

Attribute: android:label="MyApp"

Authoring XML Elements

An XML element is made up of a start tag, an end tag, and data in between.

Example:

```
<director> Matthew Dunn  </director>
```

Example of another element with the same value:

```
<actor>  Matthew Dunn </actor>
```

XML tags are case-sensitive:

```
<CITY>  <City>  <city>
```

XML can abbreviate empty elements, for example:

`<married> </married>` can be abbreviated to

```
<married/>
```

Authoring XML Elements (cont’d)

An attribute is a name-value pair separated by an equal sign (=).

Example:

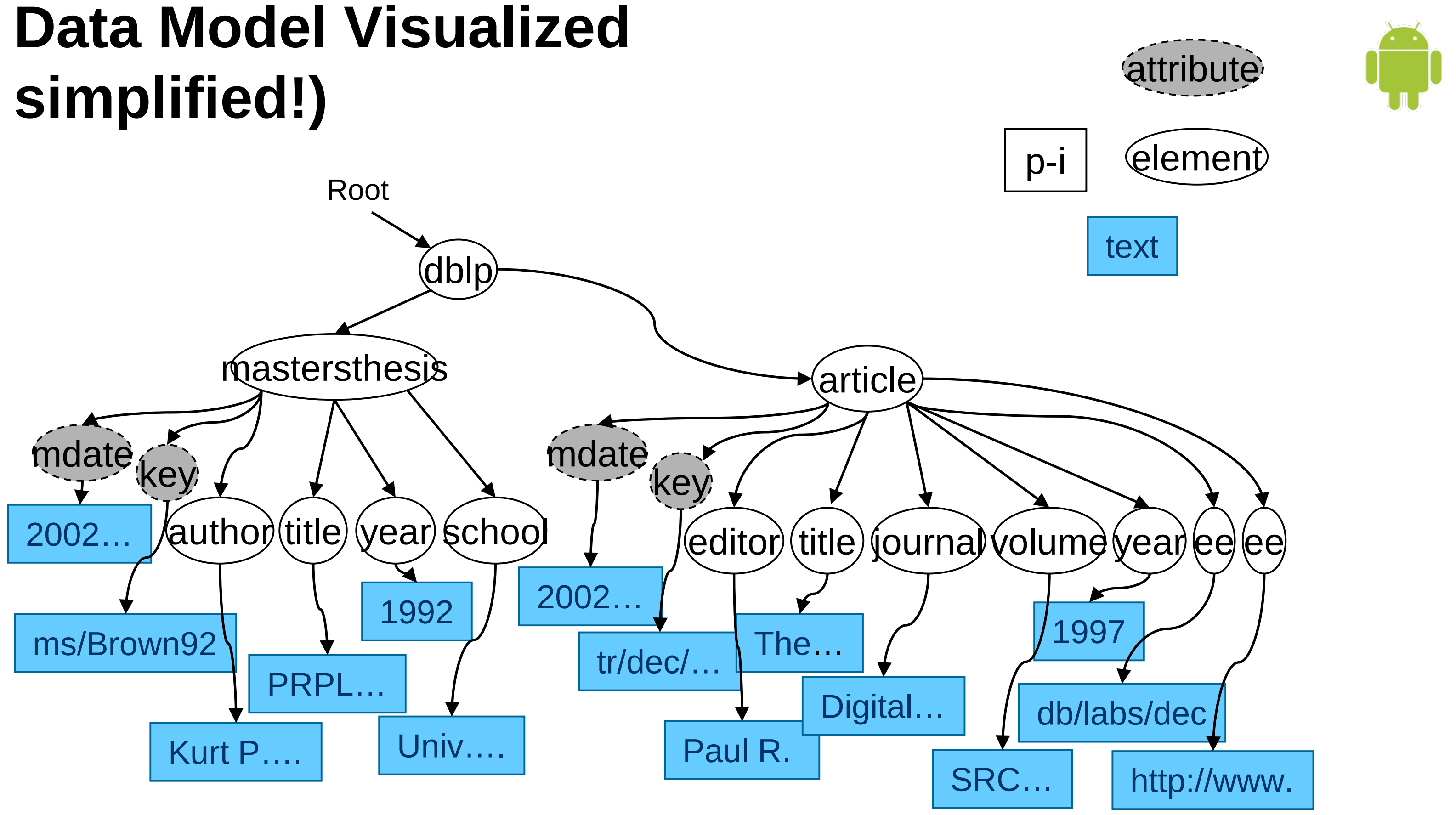
```
<City  ZIP=“94608”> Emeryville </City>
```

Attributes are used to attach additional, secondary information to an element.

XML Anatomy

```
<?xml version="1.0" encoding="ISO-8859-1" ?>          Processing Instr.
<dblp>          Open-tag
  <mastersthesis mdate="2002-01-03" key="ms/Brown92">
    <author>Kurt P. Brown</author>
    <title>PRPL: A Database Workload Specification Language</title>
    <year>1992</year>
    <school>Univ. of Wisconsin-Madison</school>          Element
  </mastersthesis>
  <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
    <editor>Paul R. McJones</editor>          Attribute
    <title>The 1995 SQL Reunion</title>
    <journal>Digital System Research Center Report</journal>
    <volume>SRC1997-018</volume>          Close-tag
    <year>1997</year>
    <ee>db/labs/dec/SRC1997-018.html</ee>
    <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
  </article>
```

XML Data Model Visualized (and simplified!)



Parsing XML - Strategies

1. **DOM (Document Object Model)** - DocumentBuilder - Produces a tree-like representation of the document. Nodes in the tree are treated as familiar Java Lists.
2. **SAX (Simple API for XML)** - XmlPullParser - You traverse the document programmatically looking for the beginning and ending of element tags, their associated text and internal attributes.
3. Many other parsers exist....

DOM Parser - DocumentBuilder

DocumentBuilder

The parser will create Java-like lists to store all the text and attributes held in each node type.

<Elements> from the input XML file become nodes in an **internal tree representation** of the dataset. The node labeled <Document> acts as the root of the tree.

DocumentBuilder

PHASE 1. For each XML element you can request a NodeList using the getElementsByTagName() method.

PHASE 2. Explore an individual node from a NodeList using the methods:

- item(i)
- getNodeName()
- getNodeValue()
- getFirstChild()
- getAttributes()
- etc.

DocumentBuilder - Simple Example

```
// Normally the input stream is from a file or the internet,
// but we define it inline here for demo purposes.
// <foo id="123">Hello World!</foo>
String xml = "<foo id=\"123\">Hello World!</foo>";
InputStream stream = new ByteArrayInputStream(xml.getBytes("UTF-8"));

// Boilerplate code to set up the parser
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(stream);
doc.getDocumentElement().normalize();
```

DocumentBuilder - Simple Example

```
// Analyze the doc
NodeList nodeList = doc.getElementsByTagName("foo");

// Analyze the first element
Node item = nodeList.item(0);
System.out.println("Element name: " + item.getNodeName());
System.out.println("Text content: " + item.getTextContent());

// Analyze the first attribute
Node attribute = item.getAttributes().item(0);
System.out.println("Attribute name: " + attribute.getNodeName());
System.out.println("Attribute value: " + attribute.getNodeValue());
```

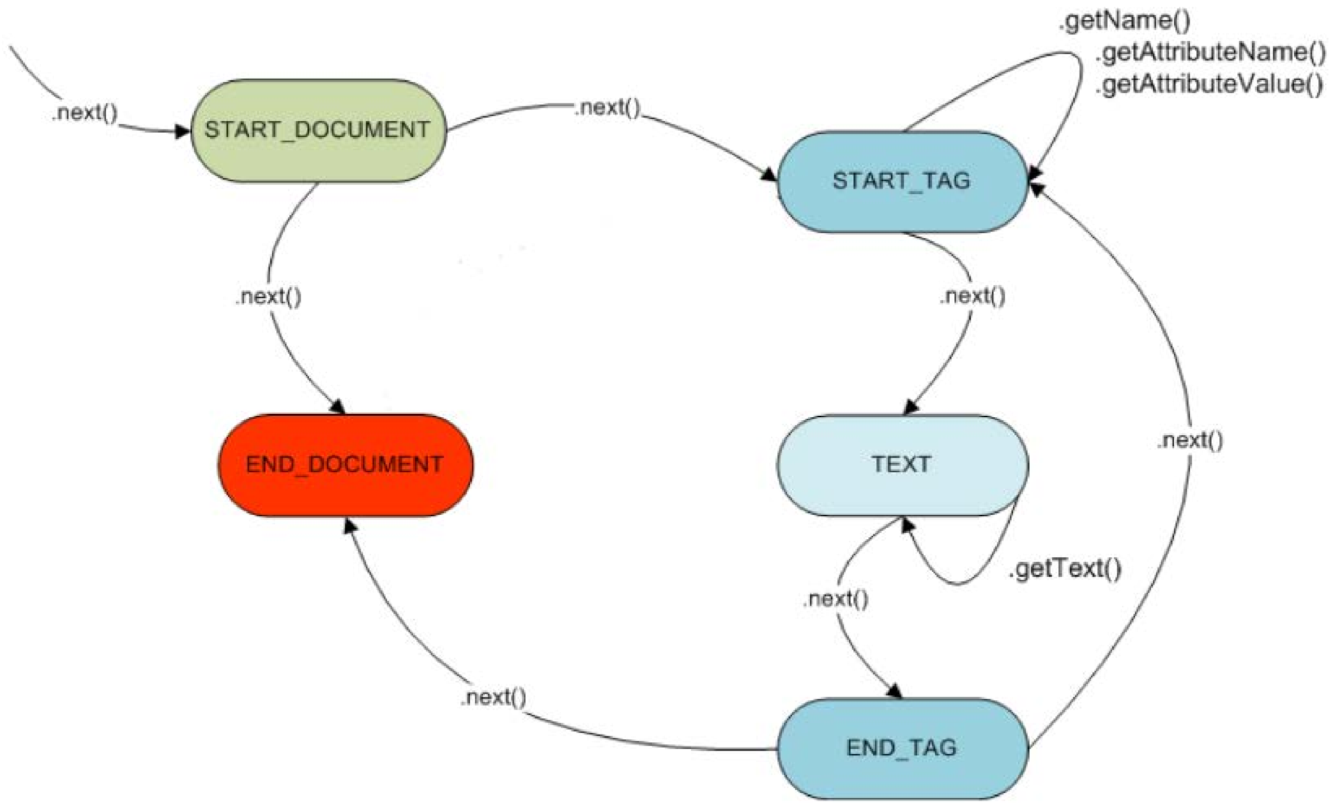
SAX Parser - XmlPullParser

SAX XmlPullParser

SAX XmlPullParser will traverse the document using the `next()` method to detect the following main eventTypes:

- `START_TAG` - An XML start tag was read.
- `TEXT` - Text content was read
- `END_TAG` - An end tag was read
- `END_DOCUMENT` - No more events are available

Document Lifecycle



Common methods

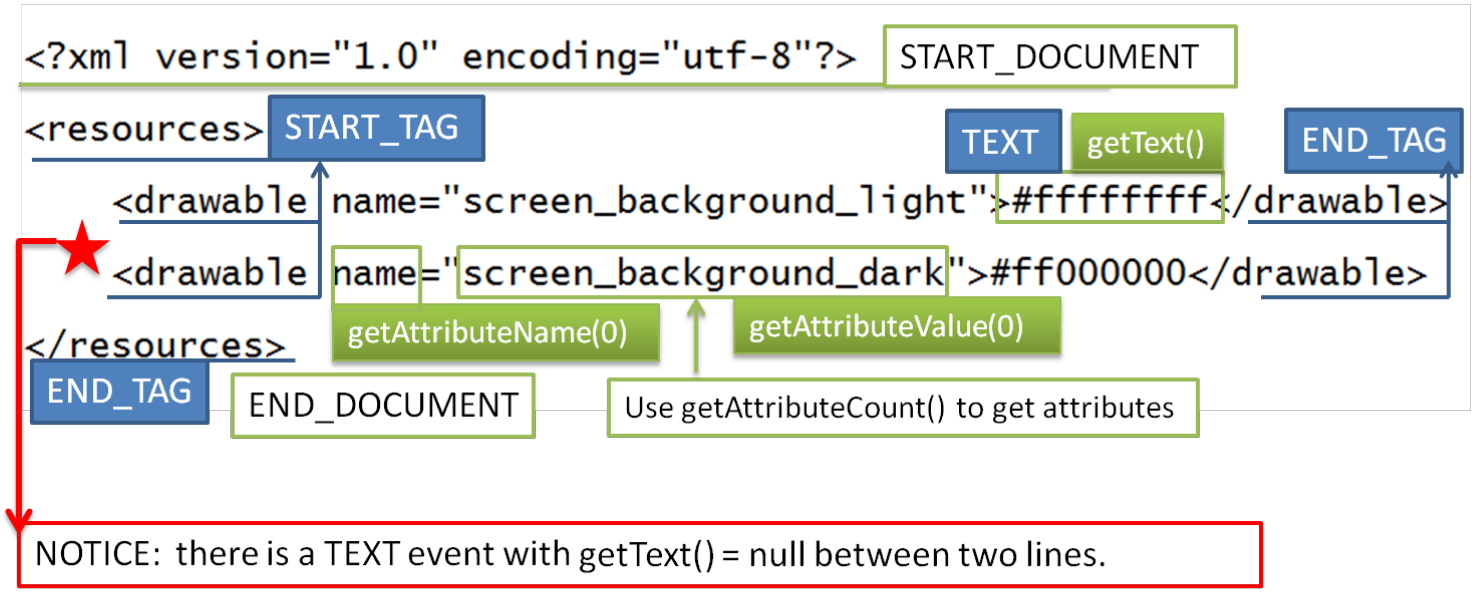
At START_TAG event:

- `getName()`- returns element name
- `getAttributeName(index)` - returns attribute name at position i
- `getAttributeValue(index)` - returns attribute value at position i

At TEXT event:

- getText() - to extract data after TEXT event

SAX XmlPullParser



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <drawable name="screen_background_light">#ffffffff</drawable>
    <drawable name="screen_background_dark">#ff000000</drawable>
</resources>
```

NOTICE: there is a TEXT event with getText() = null between two lines.

Simple XML Example

```
String document = "<foo>Hello World!</foo>";

XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
factory.setNamespaceAware(true);
XmlPullParser xpp = factory.newPullParser();
xpp.setInput(new StringReader(document));

int eventType = xpp.getEventType();
while (eventType != XmlPullParser.END_DOCUMENT) {
          if (eventType == XmlPullParser.START_DOCUMENT) {
                    System.out.println("Start document");
          } else if (eventType == XmlPullParser.END_DOCUMENT) {
                    System.out.println("End document");
          } else if (eventType == XmlPullParser.START_TAG) {
                    System.out.println("Start tag: " + xpp.getName());
          } else if (eventType == XmlPullParser.END_TAG) {
                    System.out.println("End tag: " + xpp.getName());
          } else if (eventType == XmlPullParser.TEXT) {
                    System.out.println("Text: " + xpp.getText());
          }
          eventType = xpp.next();
}
```

$\longrightarrow$ Output:

```
Start document
Start tag: foo
Text: Hello World!
End tag: foo
```

XML Example

Practical Example: Parsing RSS feed

- Shows extracting entries (title, link, summary) from a document.

SAX vs. DOM

	SAX	DOM
Ease of use		Winner
Speed	Winner	
Memory usage	Winner	

JSON

JSON

JSON (JavaScript Object Notation) is a plain-text formatting protocol for encoding and decoding hierarchically structured data.

- Based on JavaScript
- Language and platform independent

JSON

- Based on two common programming constructs: simple arrays and objects.
- Each object is represented as an associative-array holding a collection of attributes and their values.
- An attribute’s value could be a simple data-type or another nested JSON object.

JSON example

"JSON" stands for "JavaScript Object Notation"

Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs

Example (http://secretgeek.net/json_3mins.asp):

```
{"skillz": {
    "web":[
        { "name": "html",
          "years": 5
        },
        { "name": "css",
          "years": 3
        }]
    "database":[
        { "name": "sql",
          "years": 7
        }]
}}
```

JSON syntax

An *object* is an unordered set of name/value pairs

The pairs are enclosed within braces, { }

There is a colon between the name and the value

Pairs are separated by commas

Example: { "name": "html", "years": 5 }

An *array* is an ordered collection of values

The values are enclosed within brackets, [ ]

Values are separated by commas

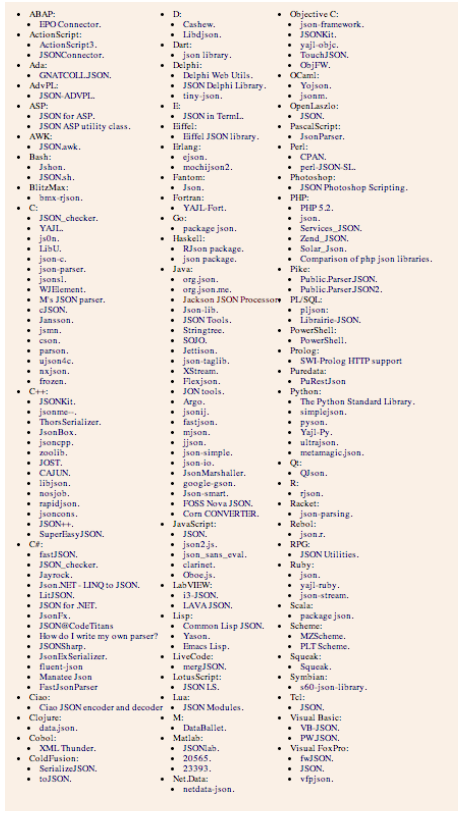
Example: ["html", ”xml", "css"]

Example

```
"Persons" : [
 {"name":"Maria", "age":21},
 {"name":"Peter", "age":33},
 {"name":"Arbie", "age":11}
]
```

JSON Parsers

- Many parsers are available (see http://www.json.org/)
- Android SDK includes org.json



- ABAP:
  - EPO Connector.
- ActionScript:
  - ActionScript3.
  - JSONConnector.
- Ada:
  - GNATCOLL.JSON.
- AdvPL:
  - JSON-ADVPL.
- ASP:
  - JSON for ASP.
  - JSON ASP utility class.
- AWK:
  - JSON.awk.
- Bash:
  - Jshon.
  - JSON.sh.
- BlitzMax:
  - bmx-rjson.
- C:
  - JSON_checker.
  - YAJL.
  - js0n.
  - LibU.
  - json-c.
  - json-parser.
  - jsonsl.
  - WJElement.
  - M's JSON parser.
  - cJSON.
  - Jansson.
  - jsmn.
  - cson.
  - parson.
  - ujson4c.
  - nxjson.
  - frozen.
- C++:
  - JSONKit.
  - jsonme--.
  - ThorsSerializer.
  - JsonBox.
  - jsoncpp.
  - zoolib.
  - JOST.
  - CAJUN.
  - libjson.
  - nosjob.
  - rapidjson.
  - jsoncons.
  - JSON++.
  - SuperEasyJSON.
- C#:
  - fastJSON.
  - JSON_checker.
  - Jayrock.
  - Json.NET - LINQ to JSON.
  - LitJSON.
  - JSON for .NET.
  - JsonFx.
  - JSON@CodeTitans
  - How do I write my own parser?
  - JSONSharp.
  - JsonExSerializer.
  - fluent-json
  - Manatee Json
  - FastJsonParser
- Ciao:
  - Ciao JSON encoder and decoder
- Clojure:
  - data.json.
- Cobol:
  - XML Thunder.
- ColdFusion:
  - SerializeJSON.
  - toJSON.
- D:
  - Cashew.
  - Libdjson.
- Dart:
  - json library.
- Delphi:
  - Delphi Web Utils.
  - JSON Delphi Library.
  - tiny-json.
- E:
  - JSON in TermL.
- Eiffel:
  - Eiffel JSON library.
- Erlang:
  - ejson.
  - mochijson2.
- Fantom:
  - Json.
- Fortran:
  - YAJL-Fort.
- Go:
  - package json.
- Haskell:
  - RJson package.
  - json package.
- Java:
  - org.json.
  - org.json.me.
  - Jackson JSON Processor.
  - Json-lib.
  - JSON Tools.
  - Stringtree.
  - SOJO.
  - Jettison.
  - json-taglib.
  - XStream.
  - Flexjson.
  - JON tools.
  - Argo.
  - jsonij.
  - fastjson.
  - mjson.
  - jjson.
  - json-simple.
  - json-io.
  - JsonMarshaller.
  - google-gson.
  - Json-smart.
  - FOSS Nova JSON.
  - Corn CONVERTER.
- JavaScript:
  - JSON.
  - json2.js.
  - json_sans_eval.
  - clarinet.
  - Oboe.js.
- LabVIEW:
  - i3-JSON.
  - LAVA JSON.
- Lisp:
  - Common Lisp JSON.
  - Yason.
  - Emacs Lisp.
- LiveCode:
  - mergJSON.
- LotusScript:
  - JSON LS.
- Lua:
  - JSON Modules.
- M:
  - DataBallet.
- Matlab:
  - JSONlab.
  - 20565.
  - 23393.
- Net.Data:
  - netdata-json.
- Objective C:
  - json-framework.
  - JSONKit.
  - yajl-objc.
  - TouchJSON.
  - ObjFW.
- OCaml:
  - Yojson.
  - jsonm.
- OpenLaszlo:
  - JSON.
- PascalScript:
  - JsonParser.
- Perl:
  - CPAN.
  - perl-JSON-SL.
- Photoshop:
  - JSON Photoshop Scripting.
- PHP:
  - PHP 5.2.
  - json.
  - Services_JSON.
  - Zend_JSON.
  - Solar_Json.
  - Comparison of php json libraries.
- Pike:
  - Public.Parser.JSON.
  - Public.Parser.JSON2.
- PL/SQL:
  - pljson:
  - Librairie-JSON.
- PowerShell:
  - PowerShell.
- Prolog:
  - SWI-Prolog HTTP support
- Puredata:
  - PuRestJson
- Python:
  - The Python Standard Library.
  - simplejson.
  - pyson.
  - Yajl-Py.
  - ultrajson.
  - metamagic.json.
- Qt:
  - QJson.
- R:
  - rjson.
- Racket:
  - json-parsing.
- Rebol:
  - json.r.
- RPG:
  - JSON Utilities.
- Ruby:
  - json.
  - yajl-ruby.
  - json-stream.
- Scala:
  - package json.
- Scheme:
  - MZScheme.
  - PLT Scheme.
- Squeak:
  - Squeak.
- Symbian:
  - s60-json-library.
- Tcl:
  - JSON.
- Visual Basic:
  - VB-JSON.
  - PW.JSON.
- Visual FoxPro:
  - fwJSON.
  - JSON.
  - vfpjson.

org.json.JSONObject

```
getString
getBoolean
getInt
getDouble
getJSONArray
```

Simple Example - Reading

```
JSONObject obj = new JSONObject(
               "{name: 'Alec', age: 50.5, misc: {hands: 2}}");
System.out.println("Name: " + obj.getString("name"));
System.out.println("Age: " + obj.getDouble("age"));
System.out.println("Hands: " +
    obj.getJSONObject("misc").getInt("hands"));
```

Simple Example - Writing

```
JSONObject obj = new JSONObject();
obj.put("name", "Michelle");
obj.put("age", 23.3);

JSONObject misc = new JSONObject();
misc.put("hands",  2);
obj.put("misc", misc);

System.out.println(obj.toString());
```

```xml
<?xml version='1.0' encoding='UTF-8'?>
<card>
   <fullname>Sean Kelly</fullname>
   <org>SK Consulting</org>
   <emailaddrs>
      <address type='work'>kelly@seankelly.biz</address>
      <address type='home' pref='1'>kelly@seankelly.tv</address>
   </emailaddrs>
   <telephones>
      <tel type='work' pref='1'>+1 214 555 1212</tel>
      <tel type='fax'>+1 214 555 1213</tel>
      <tel type='mobile'>+1 214 555 1214</tel>
   </telephones>
   <addresses>
      <address type='work' format='us'>1234 Main St
         Springfield, TX 78080-1216</address>
      <address type='home' format='us'>5678 Main St
         Springfield, TX 78080-1316</address>
   </addresses>
   <urls>
      <address type='work'>http://seankelly.biz/</address>
      <address type='home'>http://seankelly.tv/</address>
   </urls>
</card>
```

Example: An address book data encoded in XML

```
{
   "fullname": "Sean Kelly",
   "org": "SK Consulting",
   "emailaddrs": [
      {"type": "work", "value": "kelly@seankelly.biz"},
      {"type": "home", "pref": 1, "value": "kelly@seankelly.tv"}
   ],
    "telephones": [
      {"type": "work", "pref": 1, "value": "+1 214 555 1212"},
      {"type": "fax", "value": "+1 214 555 1213"},
      {"type": "mobile", "value": "+1 214 555 1214"}
   ],
   "addresses": [
      {"type": "work", "format": "us",
       "value": "1234 Main StnSpringfield, TX 78080-1216"},
      {"type": "home", "format": "us",
       "value": "5678 Main StnSpringfield, TX 78080-1316"}
   ],
    "urls": [
      {"type": "work", "value": "http://seankelly.biz/"},
      {"type": "home", "value": "http://seankelly.tv/"}
   ]
}
```

Example: The same address book data encoded in JSON

Questions?

Appendix A - Connecting to the Internet

```
// Given a string representation of a URL, sets up a connection and gets
// an input stream.
// Source: http://developer.android.com/training/basics/network-ops/xml.html
private InputStream downloadUrl(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000 /* milliseconds */);
    conn.setConnectTimeout(15000 /* milliseconds */);
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    // Starts the query
    conn.connect();
    return conn.getInputStream();
}

// Another example: http://developer.android.com/training/basics/network-
ops/connecting.html
```