DEVOXX™
the java™ community conference

JSR 353 : Java API for JSON Processing

Jitendra Kotamraju

Oracle

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract.

It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.

Program Agenda

- Overview
- JAX-RS usecase
- JSR 353
- API
- Demo

JSON Overview

- JSON is a light-weight data exchange format
  - Minimal, textual and a subset of JavaScript
  - Easy for humans/machines to read and write
  - For e.g.:

```
{"name":"Bob", "age":20, "phone":["276 1234", "123 4567"]}
```

  - Used heavily in RESTful web services, configuration, databases, browser/server communication

JSON Overview

- JSON is used by popular web sites in their RESTful web services
  - Facebook, Twitter, Amazon, …
  - Twitter Streaming API discontinues XML

JSON Overview

http://search-domainname-domainid.us-east-1.cloudsearch.amazonaws.com/2011-02-01/search?q=star+wars

```
{
    "rank":"-text_relevance",
    "match-expr":"(label 'star wars')",
    "hits":{
        "found":7,
        "start":0,
        "hit":[
            {"id":"tt0086190"},
            {"id":"tt0120915"},
            {"id":"tt0121766"}, …]
    },
    …
}
```

Source: http://docs.amazonwebservices.com/cloudsearch/latest/developerguide/searching.html

JSON Overview

```
http://search.twitter.com/search.json?q=JSON

{
    "created_at":"Thu, 06 Sep 2012 21:45:04 +0000",
    "from_user":"loggly”,
    "metadata":{"result_type":"recent"},
    "text":"Good news if you log JSON.  (And another reason to
   switch to JSON if you haven't already.) http:\/\/t.co\/
   9Dz2JP41",
    …
}
```

Source: https://dev.twitter.com/docs/using-search

JAX-RS XML usage

- JAX-RS applications handle XML using JAXP API

```
@Produces("application/xml")
public Source getBook(String id) {
    return new StreamSource(…);
}
```

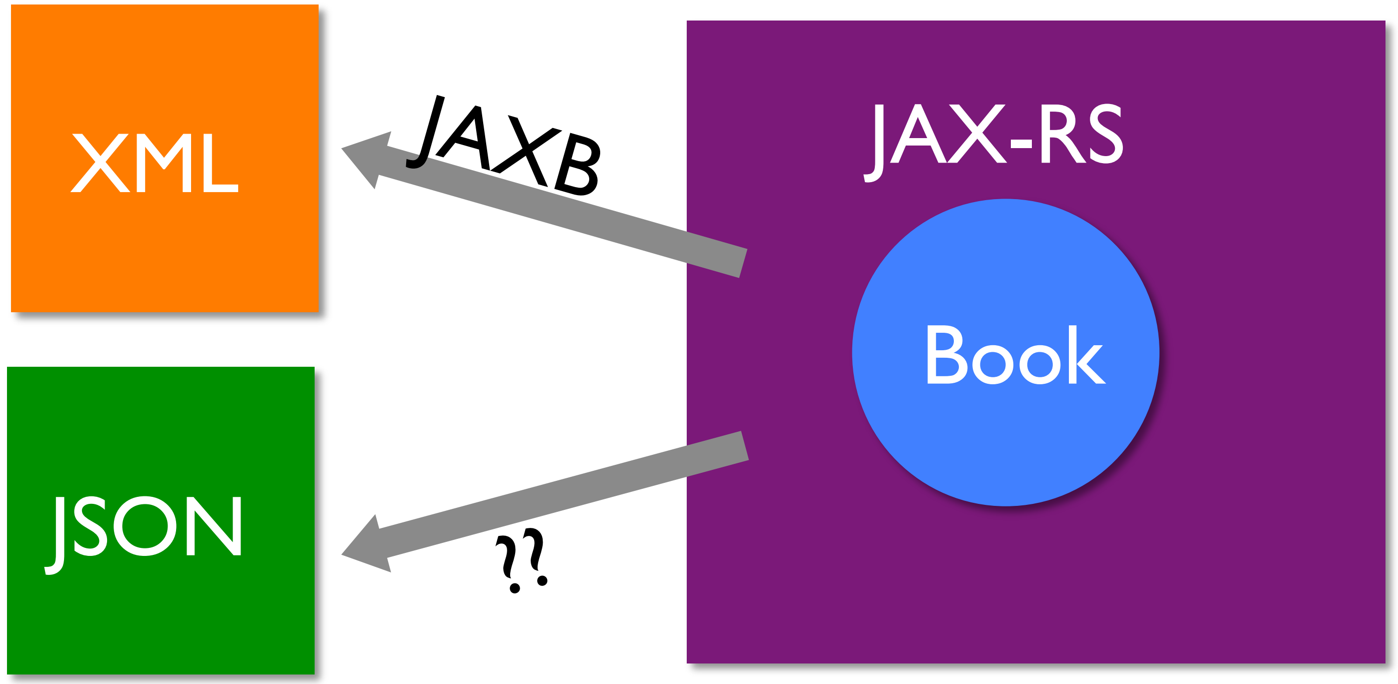
JAX-RS – XML Usage

- JAX-RS applications handle XML using JAXB API

```
@Produces("application/xml")
public Book getBook(String id) {
    return new Book(...);
}
```

JAX-RS – Content Negotiation

```
@Produces({"application/xml", "application/json"})
public Book getBook(String id) {
    return new Book(…);
}
```



Java Implementations for JSON

org.json

json-taglib

Jackson

json-simple

mjson

fastjson

jsonij

jjson

Argo

org.json.me

json-io

Json-lib

Stringtree

XStream

Jettison

SOJO

Flexjson

JsonMarshaller

google-gson

Json-smart

JAX-RS – JSON Solutions

- A custom MessageBodyWriter that converts to JSON
  - JSONObject (For e.g. json.org's API) → JSON
  - POJO/JAXB → XML → JSON (For e.g. using jettison)
  - POJO/JAXB → JSON (For e.g. using jackson, eclipseLink etc.)
- No standard API
- Some solutions have technical limitations
- Applications/Frameworks need to bundle the libraries

Standard API

- Application can use standard types
- Leaner, portable applications

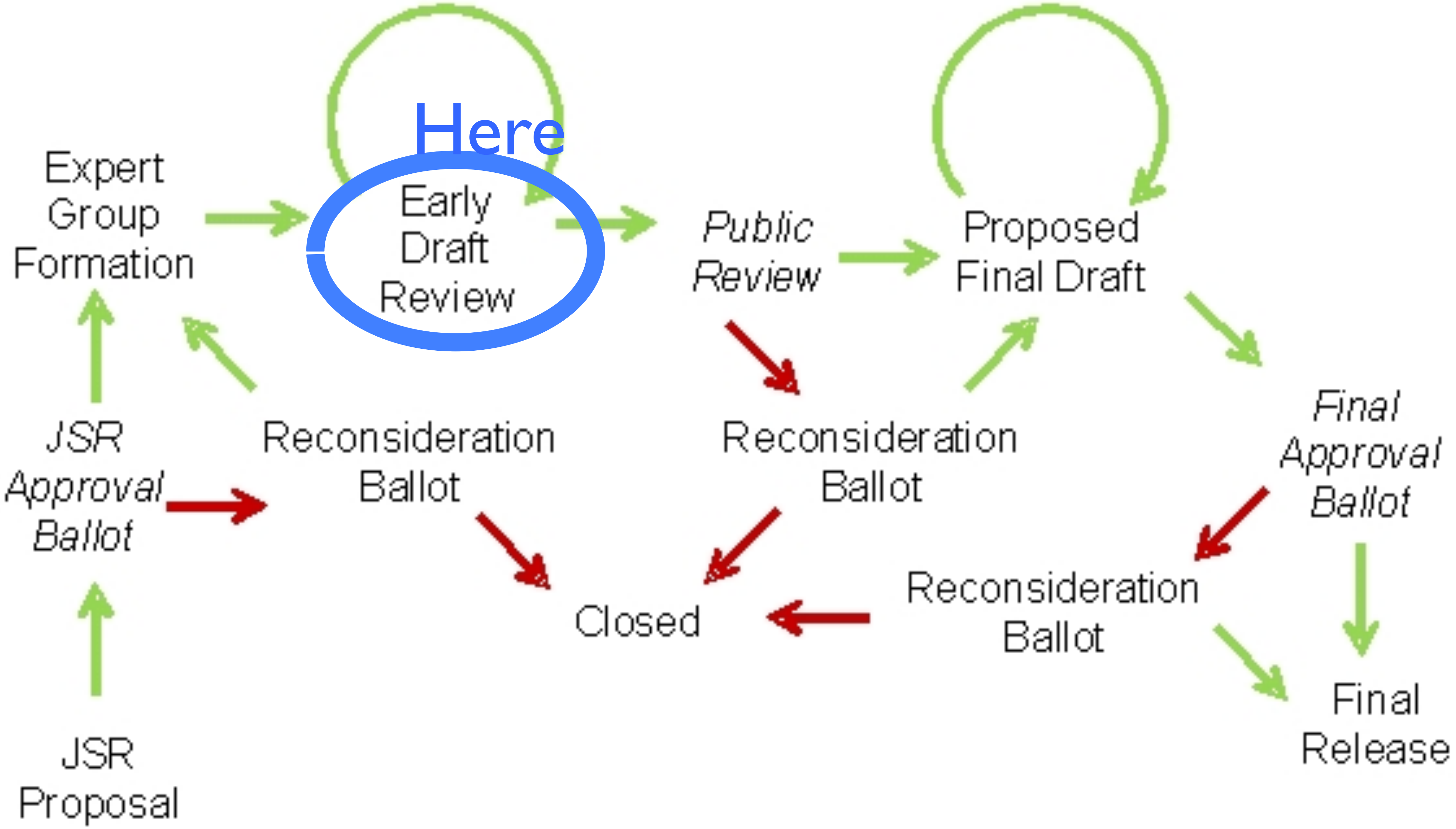
Standard API

- Parsing/Processing JSON
  - Similar to JAXP
- Data binding : JSON text <-> Java Objects
  - Similar to JAXB
- Two JSRs:
  - Processing/Parsing, Binding

JSR 353 :Java API for Processing JSON

- Streaming API to produce/consume JSON
    - Similar to StAX API in XML world
- Object model API to represent JSON
    - Similar to DOM API in XML world
- EG
    - Oracle, RedHat, Twitter
    - 3 individual members
    - And, user community !

JSR 353 - Status



Source: http://blogs.oracle.com/darcy/entry/pictorial_jcp

JSR 353 - Transparency

- Using json-processing-spec java.net open source project
- Mailing lists:
  - users@json-processing-spec.java.net
  - jsr353-experts@json-processing-spec.java.net
  - Lists are archived (publicly readable)
- Issue Tracker:
  - http://java.net/jira/browse/JSON_PROCESSING_SPEC

Source: http://www.flickr.com/photos/benjamin_galle/3142187694/

JSR 353 - Schedule

- Align with Java EE 7 schedule
  - Early Draft – Sep 2012
  - Public Review – Dec 2012
  - Proposed Final Draft – Mar 2013
  - Final Release – Apr 2013

Source:http://www.flickr.com/photos/29254399@N08/3187186308

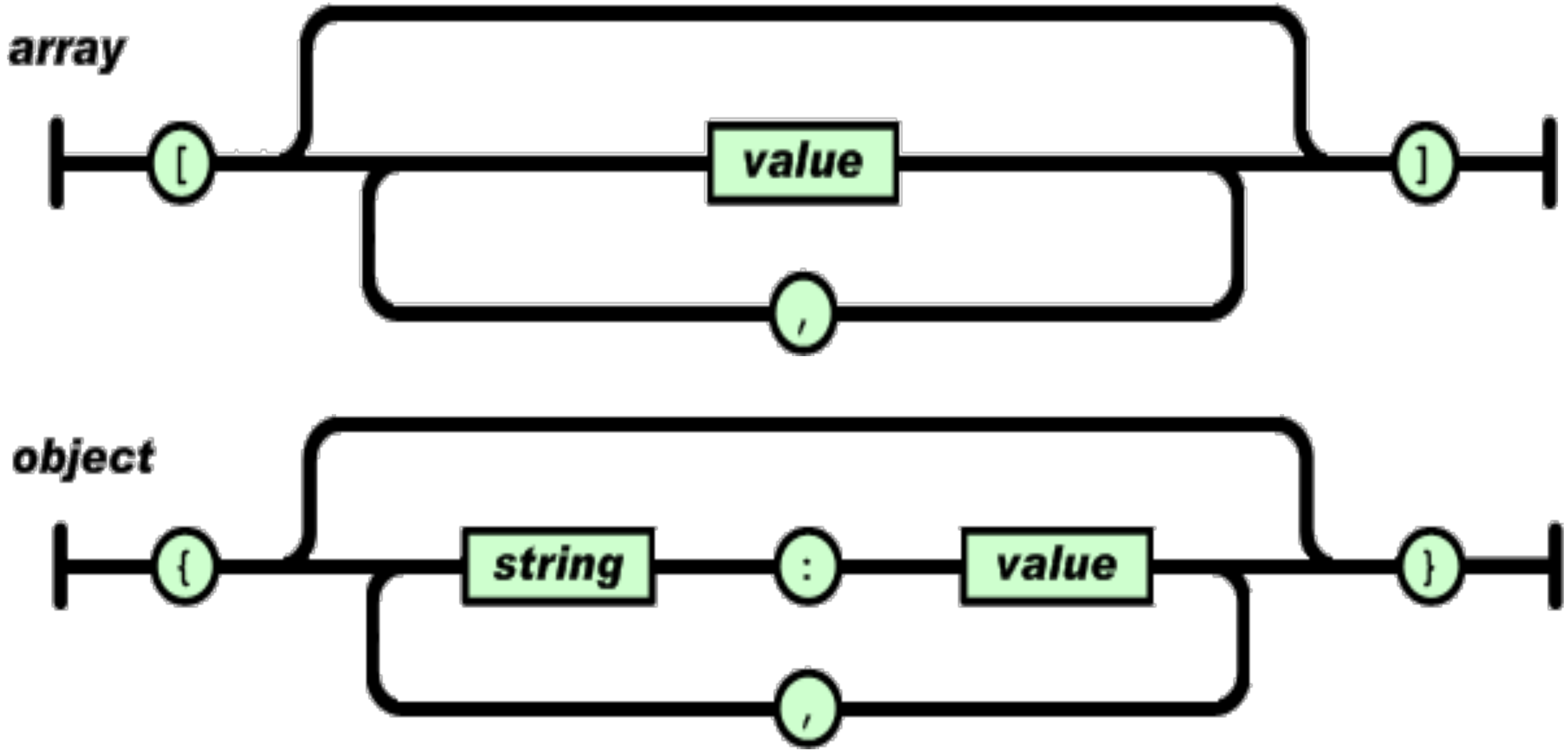
JSR 353 - RI

- Using jsonp java.net open source project
  - by GlassFish community
- Up-to-date w.r.t spec (pretty much !)
- EDR bits are in maven central

JSR 353 - Work in Progress

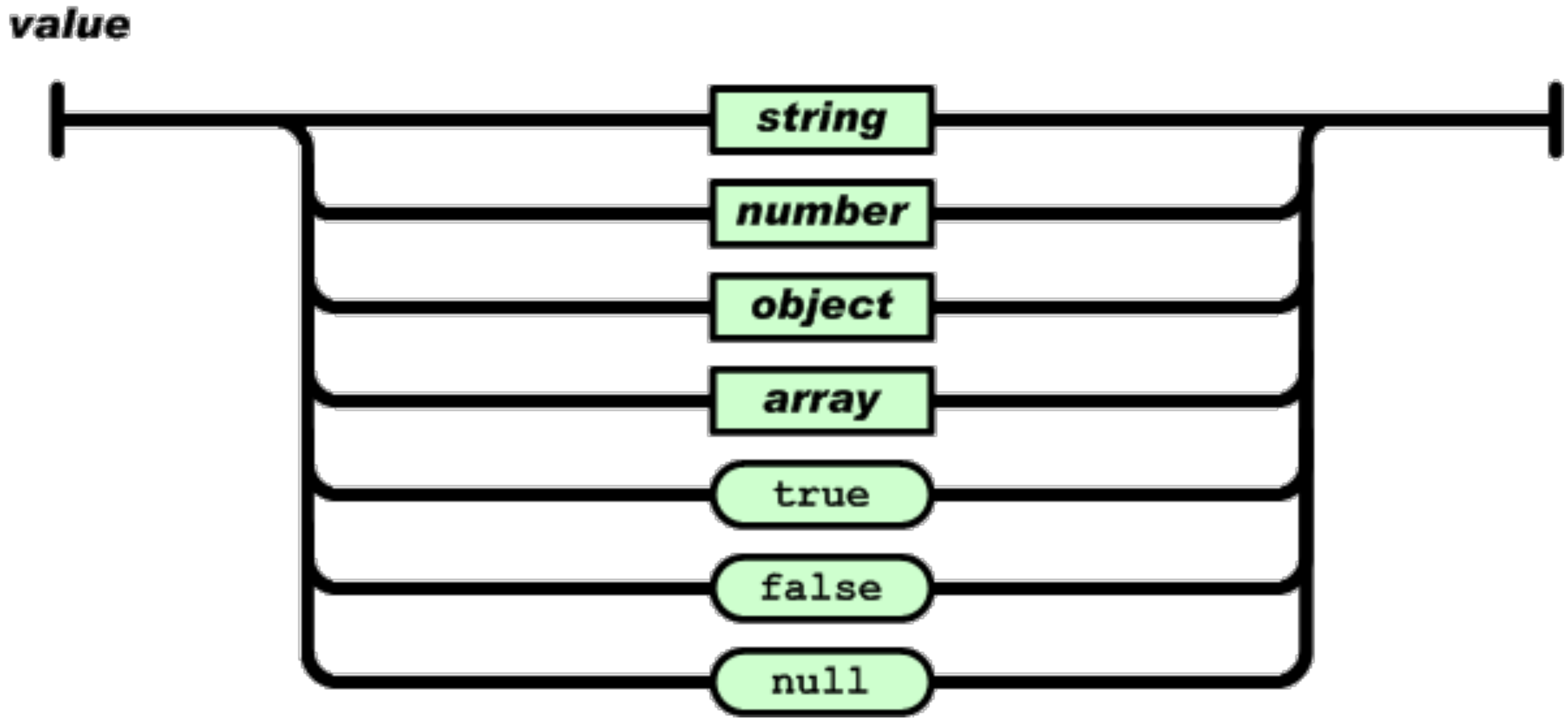
Note: The APIs might change before final release !

API - JSON Grammar



Source: http://json.org

API - JSON Grammar

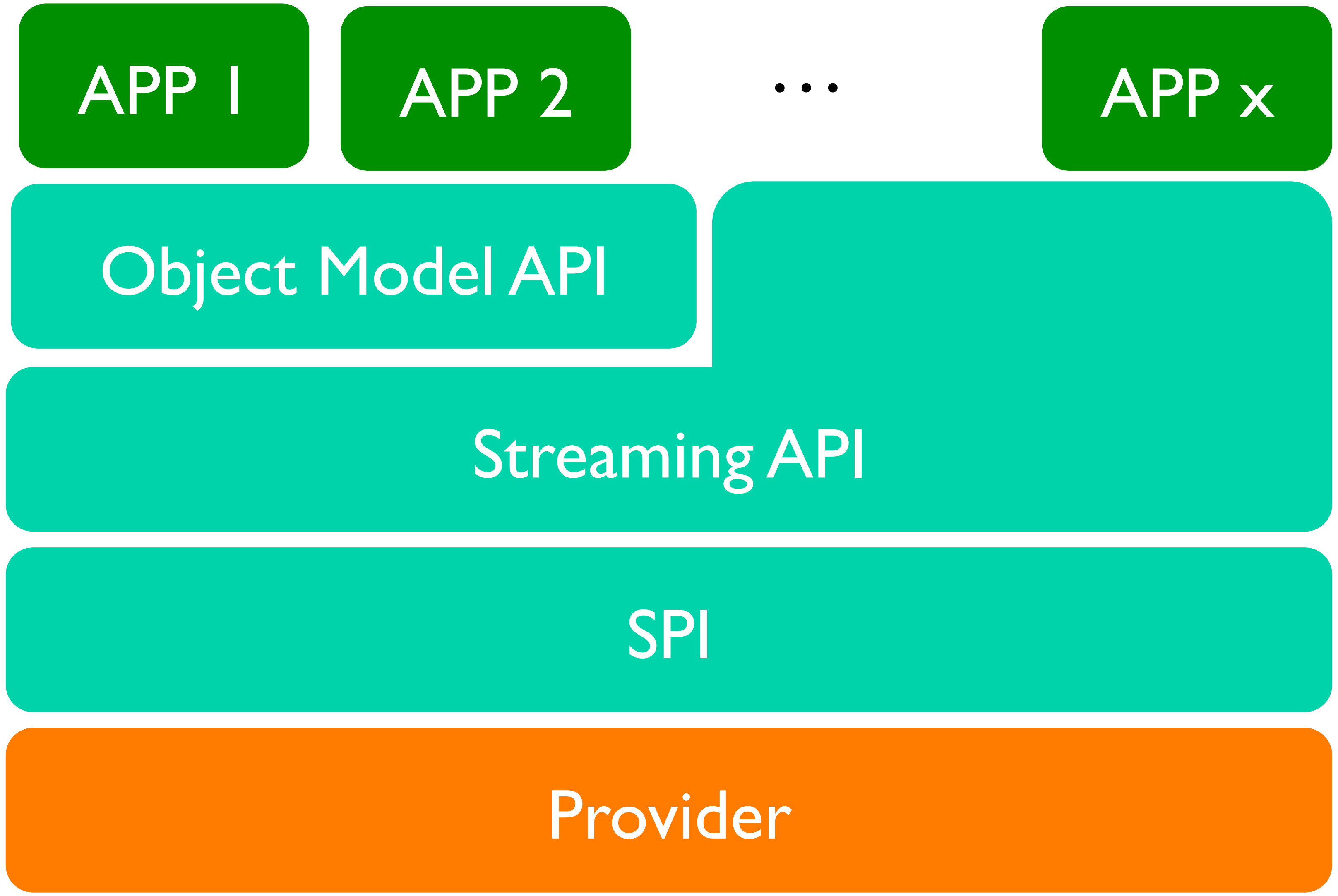


Source: http://json.org

API - Streaming & Object Model

- Streaming API
  - Low-level, efficient way to parse/generate JSON
  - Provides pluggability for parsers/generators
- Object Model API
  - Simple, easy to use high-level API
  - Implemented on top of streaming API

API Architecture



Streaming API - JsonParser

- JsonParser – Parses JSON in a streaming way from input sources
  - Similar to StAX's XMLStreamReader, a pull parser
- Created using :
  - Json.createParser(…), Json.createParserFactory().createParser(…)
- Optionally, configured with features
- Parser state events :
  - START_ARRAY, START_OBJECT, KEY_NAME, VALUE_STRING, VALUE_NUMBER, VALUE_TRUE, VALUE_FALSE, VALUE_NULL, END_OBJECT, END_ARRAY

Streaming API - JsonParser

```
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Streaming API - JsonParser

```
  START_OBJECT
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Streaming API - JsonParser

```
{                     ↑ KEY_NAME
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Streaming API - JsonParser

```
{                                 ↑ VALUE_STRING
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Streaming API - JsonParser

VALUE_NUMBER

```
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Streaming API - JsonParser

```
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

↑ START_ARRAY

Streaming API - JsonParser

```
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

END_ARRAY ↑

Streaming API - JsonParser

```
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
Iterator<Event> it = parser.iterator();
Event event = it.next();                    // START_OBJECT
event = it.next();                          // KEY_NAME
event = it.next();                          // VALUE_STRING
String name = parser.getString();           // "John”
```

Streaming API - JsonGenerator

- JsonGenerator – Generates JSON in a streaming way to output sources
  - Similar to StAX's XMLStreamWriter
- Created using :
  - Json.createGenerator(…), Json.createGeneratorFactory().createGenerator(…)
- Optionally, configured with features
  - For e.g. pretty printing
- Allows method chaining

Streaming API - JsonGenerator

```
JsonGenerator ge=Json.createGenerator(…);
ge.startWriteArray()
    .startWriteObject()
      .write("type", "home”)
      .write("number", "212 555-1234")
    .end()
    .startWriteObject()
      .write("type", "fax”)
      .write("number", "646 555-4567")
    .end()
  .end()
.close();
```

```
[
  {
    "type": "home”,
    "number": "212 555-1234"
  },
  {
    "type": "fax”,
    "number": "646 555-4567"
  }
]
```

Object Model API

- JsonObject/JsonArray – JSON object and array structures
    - JsonString and JsonNumber for string and number values
- JsonBuilder – Builds JsonObject and JsonArray
- JsonReader – Reads JsonObject and JsonArray from input source
- JsonWriter – Writes JsonObject and JsonArray to output source

Object Model API - JsonObject

- Holds name/value pairs and immutable
- Name/value pairs can be accessed as Map<String, JsonValue>

```
JsonObject obj = …;
Map<String, JsonValue> map = obj.getValues();    // as a map

String str = obj.getStringValue(“foo”);

Set<String> names = obj.getNames();   // all names
```

Object Model API - JsonObject

- Holds name/value pairs and immutable
- Name/value pairs can be accessed as Map<String, JsonValue>

```
JsonObject obj = …;
Map<String, JsonValue> map = obj.getValues();    // as a map

String str = obj.getStringValue("foo”);

Set<String> names = obj.getNames();   // all names
```

Object Model API - JsonObject

- Holds name/value pairs and immutable
- Name/value pairs can be accessed as Map<String, JsonValue>

```
JsonObject obj = …;
Map<String, JsonValue> map = obj.getValues();    // as a map

String str = obj.getStringValue("foo”);

Set<String> names = obj.getNames();   // all names
```

Object Model API - JsonObject

- Holds name/value pairs and immutable
- Name/value pairs can be accessed as Map<String, JsonValue>

```
JsonObject obj = …;
Map<String, JsonValue> map = obj.getValues();    // as a map

String str = obj.getStringValue(“foo”);

Set<String> names = obj.getNames();   // all names
```

Object Model API - JsonArray

- Holds a list of values and immutable
- Values can be accessed as List<JsonValue>

```
JsonArray arr = …;

List<JsonValue> list = arr.getValues();   // as a list

String str = arr.getStringValue(0);
```

Object Model API - JsonArray

- Holds a list of values and immutable
- Values can be accessed as List<JsonValue>

```
JsonArray arr = …;

List<JsonValue> list = arr.getValues();   // as a list

String str = arr.getStringValue(0);
```

Object Model API - JsonArray

- Holds a list of values and immutable
- Values can be accessed as List<JsonValue>

```
JsonArray arr = …;

List<JsonValue> list = arr.getValues();   // as a list

String str = arr.getStringValue(0);
```

Object Model API - JsonBuilder

- Builder to build JsonObject and JsonArray from scratch
- Allows method chaining
- Type-safe (cannot mix array and object building methods)
- Can also use existing JsonObject and JsonArray in a builder

```
// builds empty JSON object
JsonObject obj = new JsonBuilder().beginObject().endObject().build();
```

Object Model API - JsonBuilder

```
JsonArray arr = new JsonBuilder()
  .beginArray()
    .beginObject()
      .add("type", "home”)
      .add("number", "212 555-1234")
    .endObject()
    .beginObject()
      .add("type", "fax”)
      .add("number", "646 555-4567")
    .endObject()
  .endArray()
.build();
```

```
[
  {
    "type": "home”,
    "number": "212 555-1234"
  },
  {
    "type": "fax”,
    "number": "646 555-4567"
  }
]
```

Object Model API - JsonBuilder

```
//Build using existing objects/subtrees
JsonObject home = …;
JsonObject fax = …;
JsonArray arr = new JsonBuilder()
  .beginArray()
    .add(home)
    .add(fax)
  .endArray()
.build();
```

Object Model API - JsonReader

- Reads JsonObject and JsonArray from input source
  - i/o Reader, InputStream (+ encoding)
- Optionally, configured with features
- Uses pluggable JsonParser

```
// Reads a JSON object
try(JsonReader reader = new JsonReader(io)) {
    JsonObject obj = reader.readObject();
}
```

Object Model API - JsonWriter

- Writes JsonObject and JsonArray to output source
    - i/o Writer, OutputStream (+ encoding)
- Optionally, configured with features. For e.g. pretty printing
- Uses pluggable JsonGenerator

```
// Writes a JSON object
try(JsonWriter writer = new JsonWriter(io)) {
    writer.writeObject(obj);
}
```

API - Configuration

- Configuration is a set of parser/generator features
  - Pretty Printing, Single-Quoted strings
- Supports extensibility (custom features)
- Can be used in streaming & object-model API

```
// Writes a JSON object prettily
JsonConfiguration config = new
    JsonConfiguration().withPrettyPrinting();
try(JsonWriter writer = new JsonWriter(io, config)) {
    writer.writeObject(obj);
}
```

API - Configuration (future)

- Perhaps, can have a corresponding annotation for a feature
- Can be used for injection

```
public class MyServlet extends HttpServlet {
    @Custom(a=“xxx”, b=12)
    @PrettyPrinting
    @Inject
    JsonGeneratorFactory factory;

    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        factory.createGenerator(servlet.getOutputStream());
        …
    }
}
```

API - TODO

- Added Exceptions, equals()/hashCode() semantics after EDR
- Reworked JsonGenerator abstraction after EDR
- Miscellaneous

API Summary

- API provides :
  - Parsing input streams into immutable objects or event streams
  - Writing event streams or immutable objects to output streams
  - Programmatically navigating immutable objects
  - Programmatically building immutable objects with builders
- API becomes
  - A base for building data binding, transformation, querying, or other manipulation APIs

DEMO

Resources

- http://json-processing-spec.java.net
- http://jsonp.java.net
- users@json-processing-spec.java.net

Q&A