AnswerHub | Social Q&A for the Enterprise

½ of the Top 10 StackExchange 1.0 Sites Now Run on AnswerHub

Discover Why Now!

CONTENTS INCLUDE:

Core JSON

The Fat-Free Alternative to XML

By Tom Marrs

JSON OVERVIEW

JSON (JavaScript Object Notation) is a standard text-based data interchange format that enables applications to exchange data over a computer network. Programs written in Ruby, Java/EE, JavaScript, C#/.Net, PHP, etc. can easily consume and produce JSON data because it is independent of languages and computing platforms. The abundance of JSON-related APIs and tools make it easy to use JSON from your favorite programming language, IDE, and runtime environment. Additionlly, popular NoSQL databases such as MongoDB and CouchBase are based on JSON.

JSON was created by Douglas Crockford in 2001, and is specified in RFC 4627 with the IETF (Internet Engineering Task Force) standard; see http://tools.ietf.org/html/rfc4627. Per the specification, the JSON's IANA (Internet Assigned Numbers Authority) media type is `application/json`, and the file type is `.json`.

What is JSON?

JSON is a simple data format, and has 3 basic data structures:

- Name/Value (or Key/Value) Pair.
- Object.
- Arrays.

A valid JSON document is always surrounded with curly braces, like this:

```
{ JSON-Data }
```

Please note that some members of the JSON community use the term "string" rather than "document."

Why JSON?

JSON is gradually replacing XML as the preferred data exchange format on the internet because JSON is easy to read and its structures map to common programming concepts such as Objects and Arrays. JSON is more efficient (i.e., faster parsing and network transmission) than XML because JSON is more compact—there are no begin and end tags.

Name/Value Pair

A Name/Value pair looks like this:

```
{
  "firstName": "John"
}
```

A property name (i.e., firstName) is a string that is surrounded by double quotes. A value can be a string (as in the above example), but this is just one of several valid data types. (Please see the Data Types section for further details.) Some well-known technologies claim that they use JSON data formats, but they don't surround their strings with quotes. However, this is not valid JSON; see the JSON Validation section.

Object

An Object is a collection of unordered Name/Value pairs. The following example shows an `address` object:

```
{
  "address" : {
    "line1" : "555 Main Street",
    "city" : "Denver",
    "stateOrProvince" : "CO",
    "zipOrPostalCode" : "80202",
    "country" : "USA"
  }
}
```

An Object (in this case `address`) consists of comma-separated name/value pairs surrounded by curly braces.

Array

An Array is a collection of ordered values, and looks like this:

```
{
  "people" : [
    { "firstName": "John", "lastName": "Smith", "age": 35 },
    { "firstName": "Jane", "lastName": "Smith", "age": 32 }
  ]
}
```

Value Types

A Value (i.e., the right-hand side of a Name/Value Pair) can be one of the following:

- Object
- Array
- String
- Number
- Boolean
- null

Number

A number can be an integer or double-precision float. Here are some examples:

```
"age": 29
"cost": 299.99
"temperature": -10.5
"speed_of_light": 1.23e11
"speed_of_light": 1.23e+11
"speed_of_light": 1.23E11
"speed_of_light": 1.23E+11
```

The property name (i.e., **age**, etc.) is a string surrounded by double quotes, but the value does not have quotes. A number can be prefixed by a minus sign. The exponent portion (denoted by **e** or **E**) comes after the number value, and can have an optional plus or minus sign. Neither leading zeroes, octal, nor hexadecimal values are allowed.

AnswerHub
The Enterprise Q&A Platform
Work Smarter, Not Harder
Connect your entire organization with fast, accurate answers.
Learn More

Boolean

A Boolean in JSON can either be `true` or `false`, as follows:

```
{
  “emailValidated” : true
}
```

The property name `(emailValid)` is a string surrounded by double quotes, but the value `(true)` does not have quotes.

null

Although technically not a data type, `null` is a special value to indicate that a data element has no value. In the following example, the `age` field has no value (possibly because the user chose not to enter this information):

```
{
  “age” : null
}
```

Comments

JSON does not allow comments. Comments were originally a part of JSON, but developers misused them by putting parsing directives in comments. When Douglas Crockford saw this practice, he removed comments from JSON to preserve interoperability between computing platforms.

Style

You've probably noticed that the property names (i.e., the name on the left-hand side of the colon) use camel case. This is not a rule or standard, but is a convention prescribed in Google's JSON Style Guide at: http://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml.

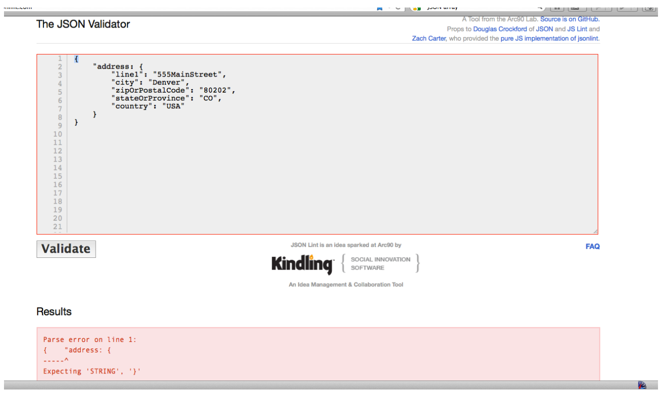
Official Syntax

Douglas Crockford's JSON site (http://www.json.org) provides a full description of JSON syntax.

Additionally, the JSON Pro Quick Guide (freely available in the iPhone App Store) provides examples and an overview of JSON syntax.

JSON VALIDATION

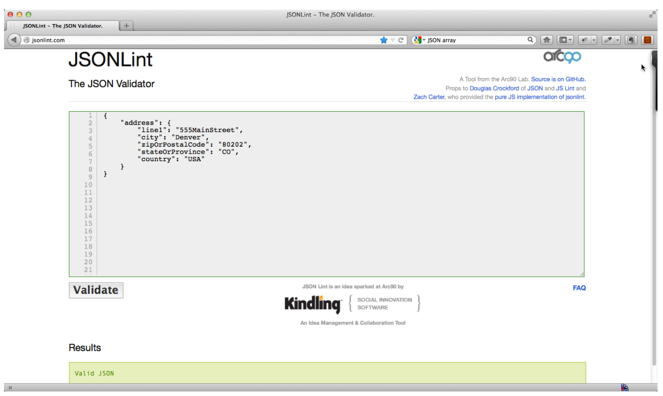
A textual document MUST follow the JSON syntax rules to be considered a valid JSON document. Valid JSON is important because it ensures interoperability between applications and JSON-related tools. Although the JSON.org web site shows JSON syntax, sometimes it's easier to see JSON validation in action. JSONLint (http://www.jsonlint.com) provides an interactive, web-based JSON validator. To use it, type or paste some text into the main text area and press the Validate button. If the text isn't valid JSON, you'll see an error message as follows:


The JSON Validator
A Tool from the Arc90 Lab. Source is on GitHub.
Props to Douglas Crockford of JSON and JS Lint and
Zach Carter, who provided the pure JS implementation of jsonlint.
Validate
JSON Lint is an idea sparked at Arc90 by
Kindling { SOCIAL INNOVATION SOFTWARE }
An Idea Management & Collaboration Tool
FAQ
Results

```
{
    "address: {
        "line1": "555MainStreet",
        "city": "Denver",
        "zipOrPostalCode": "80202",
        "stateOrProvince": "CO",
        "country": "USA"
    }
}
```

```
Parse error on line 1:
{    "address: {
-----^
Expecting 'STRING', '}'
```

In this case, the property name for the address Object is missing a closing double quote. After you fix this problem and press the Validate button, JSONLint pretty-prints the JSON document as follows:


JSONLint – The JSON Validator.
jsonlint.com
JSON array
JSONLint
The JSON Validator
arc90
A Tool from the Arc90 Lab. Source is on GitHub.
Props to Douglas Crockford of JSON and JS Lint and
Zach Carter, who provided the pure JS implementation of jsonlint.
Validate
JSON Lint is an idea sparked at Arc90 by
Kindling { SOCIAL INNOVATION SOFTWARE }
An Idea Management & Collaboration Tool
FAQ
Results

```
{
    "address": {
        "line1": "555MainStreet",
        "city": "Denver",
        "zipOrPostalCode": "80202",
        "stateOrProvince": "CO",
        "country": "USA"
    }
}
```

```
Valid JSON
```

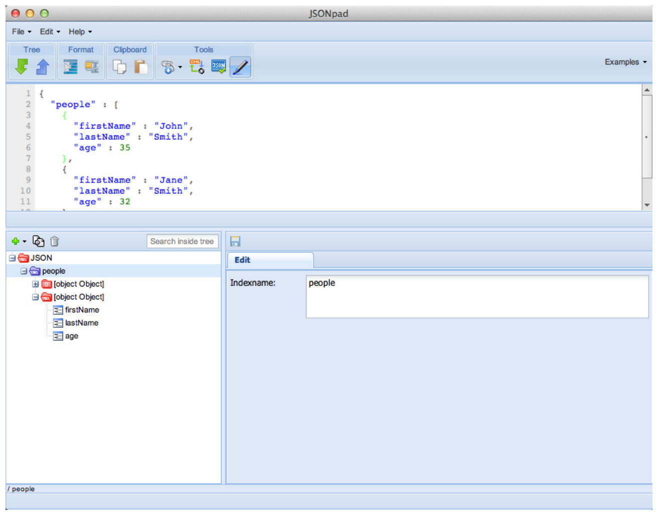
JSONLint is also available as a Chrome extension in the Chrome Web Store.

JSON MODELING

Developing valid JSON documents for real applications can be tedious and error-prone. To avoid typographical errors, you can use JSONPad, JSON Editor Online, and JSON Designer to create a logical model (similar to UML) and generate valid JSON documents.

JSONPad

JSONPad (from http://www.jsonpad.com/en/Home.html) is a GUI tool that eliminates typing JSON text by providing an interface that enables you to create Objects, Keys (i.e., Name/Value Pairs), and Arrays. JSONPad is available as a Windows or Mac GUI, and online at the JSONPad web site. To create a model, use the green plus key under the text area. The following data types are supported: Key (i.e., Name/Value Pair), Object, and Array. After the model is complete, press the blue up-arrow button (under the Tree tab) to generate a valid, pretty-printed JSON document based on the model:



```
{
  "people" : [
    {
      "firstName" : "John",
      "lastName" : "Smith",
      "age" : 35
    },
    {
      "firstName" : "Jane",
      "lastName" : "Smith",
      "age" : 32
```

Indexname: people

The end result is a valid JSON document that is usable in your application. You can also generate a model by pasting JSON text into the text area and pressing the green down arrow in the Tree tab. Under the Format tab, you can either compress or pretty print a JSON document. JSONPad validates the JSON document in the text are when you press the JSON button in the Tools tab.

JSON Editor Online

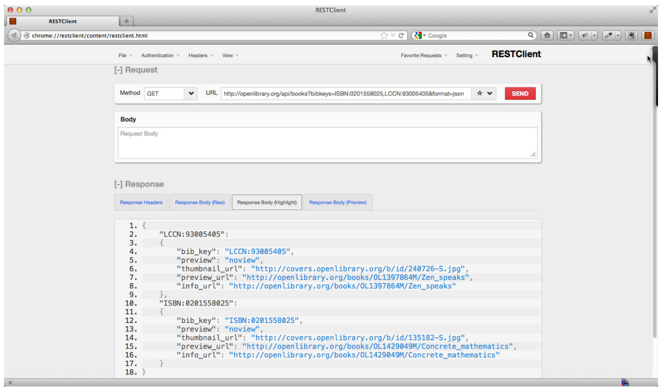
JSON Editor Online (http://jsoneditoronline.org/) is an online JSON modeler, and is also available as a Chrome extension.

JSON IN THE BROWSER

Firefox and Chrome provide excellent extensions (i.e., add-ons and plugins) that make it easier to work with JSON.

REST Client

Rest Client is a Firefox extension that provides the ability to debug and test RESTful Web Services from the browser. The ability to test from the browser isn't new, but the output formatting is much more readable.



```
{
    "LCCN:93005405":
    {
        "bib_key": "LCCN:93005405",
        "preview": "noview",
        "thumbnail_url": "http://covers.openlibrary.org/b/id/240726-S.jpg",
        "preview_url": "http://openlibrary.org/books/OL1397864M/Zen_speaks",
        "info_url": "http://openlibrary.org/books/OL1397864M/Zen_speaks"
    },
    "ISBN:0201558025":
    {
        "bib_key": "ISBN:0201558025",
        "preview": "noview",
        "thumbnail_url": "http://covers.openlibrary.org/b/id/135182-S.jpg",
        "preview_url": "http://openlibrary.org/books/OL1429049M/Concrete_mathematics",
        "info_url": "http://openlibrary.org/books/OL1429049M/Concrete_mathematics"
    }
}
```

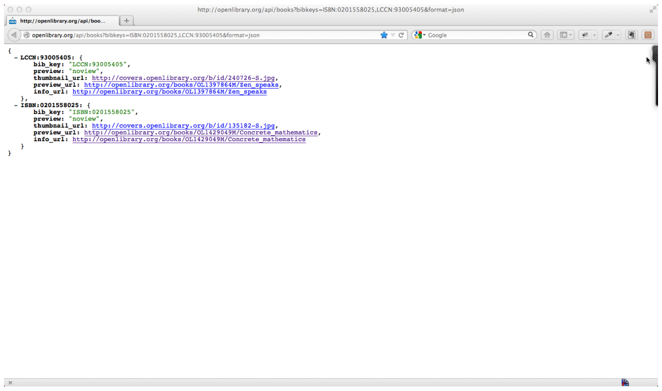
The above example uses the Books service from the Open Library API. After entering the service URI, the Response Body (Highlight) tab shows the JSON output.

Pretty Printing with JSONView

JSON is not very readable when displayed natively in a browser. JSONView is a Firefox and Chrome extension that pretty-prints JSON.

JSONView in Firefox

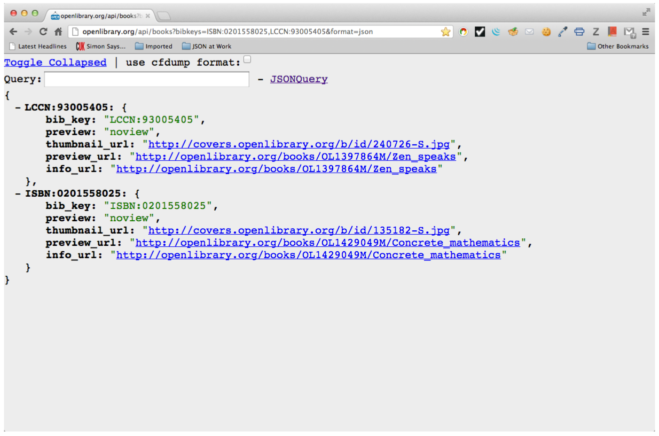
After installing this extension and re-starting Firefox, the JSON response from the Open Library Books service URI is readable, and you can expand/collapse objects on the page:


http://openlibrary.org/api/books?bibkeys=ISBN:0201558025,LCCN:93005405&format=json

```
{
  - LCCN:93005405: {
        bib_key: "LCCN:93005405",
        preview: "noview",
        thumbnail_url: http://covers.openlibrary.org/b/id/240726-S.jpg,
        preview_url: http://openlibrary.org/books/OL1397864M/Zen_speaks,
        info_url: http://openlibrary.org/books/OL1397864M/Zen_speaks
    },
  - ISBN:0201558025: {
        bib_key: "ISBN:0201558025",
        preview: "noview",
        thumbnail_url: http://covers.openlibrary.org/b/id/135182-S.jpg,
        preview_url: http://openlibrary.org/books/OL1429049M/Concrete_mathematics,
        info_url: http://openlibrary.org/books/OL1429049M/Concrete_mathematics
    }
}
```

JSONView in Chrome

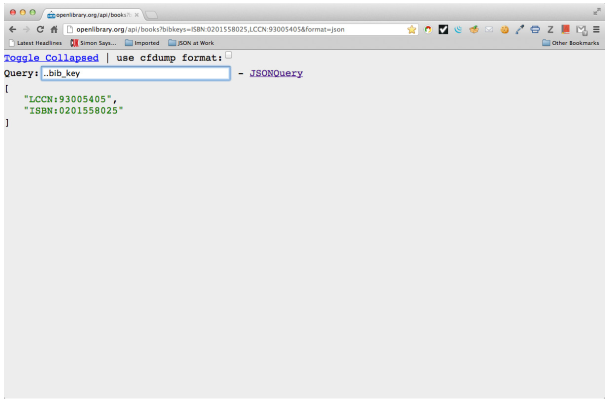
JSONView is also available as a Chrome extension from the Chrome Web Store:


openlibrary.org/api/books?bibkeys=ISBN:0201558025,LCCN:93005405&format=json
Latest Headlines
Simon Says...
Imported
JSON at Work
Other Bookmarks

```
Toggle Collapsed | use cfdump format:
Query:                              - JSONQuery
{
  - LCCN:93005405: {
        bib_key: "LCCN:93005405",
        preview: "noview",
        thumbnail_url: "http://covers.openlibrary.org/b/id/240726-S.jpg",
        preview_url: "http://openlibrary.org/books/OL1397864M/Zen_speaks",
        info_url: "http://openlibrary.org/books/OL1397864M/Zen_speaks"
    },
  - ISBN:0201558025: {
        bib_key: "ISBN:0201558025",
        preview: "noview",
        thumbnail_url: "http://covers.openlibrary.org/b/id/135182-S.jpg",
        preview_url: "http://openlibrary.org/books/OL1429049M/Concrete_mathematics",
        info_url: "http://openlibrary.org/books/OL1429049M/Concrete_mathematics"
    }
}
```

Click on the minus sign to collapse each element, and press the `Toggle Collapsed` link to show/hide each collapsed element.

The JSONView Chrome extension provides a bit more functionality than its Firefox counterpart – it enables a user to search a JSON document using the JSONQuery language. For example, entering `..bib_key` in the Query text box displays all the `bib_key` fields in the text:


openlibrary.org/api/books?bibkeys=ISBN:0201558025,LCCN:93005405&format=json
Latest Headlines
Simon Says...
Imported
JSON at Work
Other Bookmarks

```
Toggle Collapsed | use cfdump format:
Query: ..bib_key                    - JSONQuery
[
  "LCCN:93005405",
  "ISBN:0201558025"
]
```

JSONQuery is one of several technologies that can search JSON documents and return the desired element(s).

JSON Beautification with JSON SH

JSON SH is a Chrome extension (available in the Google Chrome Web Store) that acts as a pretty-printer and a validator. Paste a valid (but not pretty) JSON document into the text area at the top of the page, and JSON SH beautifies the text into a human-readable format.

JSON AND AJAX

AJAX (Asynchronous JavaScript and XML) was one of the original use cases for JSON, and the following jQuery example shows how a JavaScript client makes an HTTP Get request to a RESTful Web Service and processes a JSON response:

```
$.getJSON(‘http://example/service/addresses/home/1’,
  function(data) {
    var address = JSON.parse(data);

    console.log(“Address Line 1 = “ + address.line1);
  }
);
```

In the code above, `$.getJSON()` (a shorthand version of the main jQuery `$.ajax()` call) makes an HTTP GET request. The (anonymous) success callback function receives the JSON response and parses it into a JavaScript object using `JSON.parse()`, which is part of the ECMA-262 standard (beginning with the 5th edition) – please see http://www.ecmascript.org/ for further information). The `console.log()` method then logs line 1 of the address to the browser console. Conversely, the `JSON.stringify()` method converts a JavaScript value to a JSON string (with optional pretty-printing).

JSON AND JAVA

The Jackson (http://jackson.codehaus.org/) library is a popular Java-based JSON API. Here's an example of how to marshal/unmarshal an Address object to/from JSON:

```
import java.io.Writer;
import java.io.StringWriter;
import org.codehaus.jackson.map.ObjectMapper;

public class Address {
    private String line1;
    private String city;
    private String stateOrProvince;
    private String zipOrPostalCode;
    private String country;

    public Address() {}

    public String getLine1() {
        return line1;
    }                            <!---Snippet con’t on next page-->
    public void setLine1(line1) {
       this.line1 = line1;
    }

    // Remaining getters and setters ...
}

Address addrOut = new Address();
// Call setters to populate addrOut …

ObjectMapper mapper = new ObjectMapper(); // Reuse this.

// Marshal Address object to JSON String.
Writer writer = new StringWriter();
mapper.writeValue(writer, addrOut);
System.out.println(writer.toString());

// Unmarshal Address object from JSON String.
String addrJsonStr =
"{" +
    "\"address\" : {" +
    "\"line1\" : \"555 Main Street\"," +
    "\"city\" : \"Denver\","
    "\"stateOrProvince\" : \"CO\","
    "\"zipOrPostalCode\" : \"80202\"," +
    "\"country\" : \"USA\"" +
    "}" +
"}";

Address addrIn = mapper.readValue(addrJsonStr, Address.class);
```

In addition to Jackson, other well-known Java-based JSON APIs include:

API	Source
Google GSON	http://code.google.com/p/google-json/
SOJO	http://sojo.sourceforge.net/
org.json (by Douglas Crockford)	http://www.json.org/java
json-lib	http://sourceforge.net/projects/json-lib/
json-io	http://code.google.com/p/json-io
jsontools	http://jsontools.berlios.de/
jsonbeans	http://code.google.com/p/jsonbeans/

JSON AND RUBY

There are many JSON-related libraries for Ruby. Here's an example using the JSON gem that comes standard with Ruby.

```
require 'json'

class Address

  attr_accessor :line1, :city, :state_or_province,
                :zip_or_postal_code, :country

  def initialize(line1='', city='', state_or_province='',
                 zip_or_postal_code='', country='')
    @line1 = line1
    @city = city
    @state_or_province = state_or_province
    @zip_or_postal_code = zip_or_postal_code
    @country = country
  end

  def to_json
    to_hash.to_json
  end

  def from_json!(str)
    JSON.parse(str).each { |var, val| send("#{var}=", val) }
  end

  private

  def to_hash
    Hash[instance_variables.map { |var| [var[1..-1].to_sym,
         send(var[1..-1])] }]
  end
end
```

The JSON gem's `to_json` method converts a String or Hash to JSON. The `Address` object's `to_json` method converts an Address to JSON format by converting its data members to a Hash and then calling to `to_json` on the Hash. To convert the Address to JSON, do the following:

```
addr1 = Address.new('555 Main Street', 'Denver', 'CO', '80231',
'US')
puts addr1.to_json

# Outputs the following …
{"line1":"555 Main Street","city":"Denver","state_or_
province":"CO","zip_or_postal_code":"80231","country":"US"}
```

The JSON gem's `JSON.parse` method converts a JSON String to a Hash. The `Address` object's `from_json!` method takes a JSON String, calls `JSON.parse` to convert to a Hash, and sets each corresponding data member from the Hash as follows:

```
json_addr = <<END
{
  "line1" : "999 Broadway", "city" : "Anytown",
  "state_or_province" : "CA", "zip_or_postal_code" : "90210",
  "country" : "USA"
}
END

addr2 = Address.new
addr2.from_json!(json_addr)
```

In addition to the JSON gem, other JSON-related gems include:

API	Source
ActiveSupport JSON	http://api.rubyonrails.org/classes/ActiveSupport/JSON.html
Yajl	https://github.com/brianmario/yajl-ruby
Oj	https://github.com/ohler55/oj

JSON AND RUBY ON RAILS

Ruby on Rails provides additional functionality that makes it easier to convert Ruby objects to JSON. The following controller uses the ActionController's render method to output an Address object to JSON:

```
class Person
  attr_accessor :first_name, :last_name

  def initialize(first_name=nil, last_name=nil)
    @first_name = first_name
    @last_name = last_name
  end
end

class MyController < ApplicationController
  def index
    person = Person.new('John', 'Doe')
    respond_to do |format|
      format.html # index.html.erb
      format.json { render :json => person}
    end
  end
end
```

The Rails `ApplicationController` takes care of marshalling/unmarshalling objects to/from JSON, so there's no need to write a `to_json` method here.

JSON SCHEMA

JSON Schema specifies the structure of a JSON document. JSON Schema can be used to validate the content of JSON sent to/received from a RESTful Web Service. JSON Schemas are written in JSON.

The main JSON Schema site can be found at: http://json-schema.org. JSON Schema is a work in progress – the JSON Schema team has just published version 0.4, which can be found at: http://tools.ietf.org/html/draft-zyp-json-schema-04.

Some important JSON Schema constructs include:

Construct	Description
type	The data type object, array, string, number, etc.
$schema	The URI that provides the schema version.
required	true/false
id	Data element id
properties	Validation properties for a data element include `type` (see above), `minimum` (minimum value), `maximum` (maximum value), `enum`, etc.

Here is a sample JSON Schema for a portion of an online gift registry:

```
    “type”: “object”,
    “$schema”: “http://json-schema.org/draft-03/schema”,
    “id”: “#”,
    “required”: true,
    “properties”: {
        “registrants”: {
            “type”: “array”,
            “id”: “registrants”,
            “required”: true,
            “items”: {
                “type”: “object”,
                “required”: false,
                “properties”: {
                    “address”: {
                        “type”: “object”,
                        “id”: “address”,
                        “required”: true,
                        “properties”: {
                            “city”: {
                                “type”: “string”,
                                “id”: “city”,
                                “required”: true
                            },
                            “country”: {
                                “type”: “string”,
                                “id”: “country”,
                                “required”: false
                            },
                            “line1”: {
                                “type”: “string”,
                                “id”: “line1”,
                                “required”: true
                            },
                            “line2”: {
                                “type”: “string”,
                                “id”: “line2”,
                                “required”: false
                            },
                            “postalCode”: {
                                “type”: “string”,
                                “id”: “postalCode”,
                                “required”: true
                            },
                            “premise”: {
                                “type”: “string”,
                                “id”: “premise”,
                                “required”: true,
                                “enum”: [
                                    “work”,
                                    “home”,
                                    “other”
                                ]
                            },
                            “stateOrProvince”: {
                                “type”: “string”,
                                “id”: “stateOrProvince”,
                                “required”: true
                            }
                        }
                    },
                    “firstName”: {
                        “type”: “string”,
                        “id”: “firstName”,
                        “required”: true
                    },
                    “lastName”: {
                        “type”: “string”,
                        “id”: “lastName”,
                        “required”: true
                    },
```

<!============Snippet con't on next page---------------->

```
                    “phoneNumber”: {
                        “type”: “object”,
                        “id”: “phoneNumber”,
                        “required”: true,
                        “properties”: {
                            “channel”: {
                                “type”: “string”,
                                “id”: “channel”,
                                “required”: true,
                                “enum”: [
                                    “cell”,
                                    “work”,
                                    “home”
                                ]
                            },
                            “number”: {
                                “type”: “string”,
                                “id”: “number”,
                                “required”: true
                            }
                        }
                    }
                }
            }
        }
    }
}
```

The above schema:

- Requires an array of `registrant` objects.
- Limits the phoneNumber.`channel` field to the following values: `cell, work, fax`, or home.
- Limits the address.premise field to these values: `home, work,` or `other.`

A Web Service consumer could use this schema to validate the following JSON document:

```
{
    “registrants”: [
        {
            “firstName”: “Fred”,
            “lastName”: “Smith”,
            “phoneNumber”: {
                “channel”: “cell”,
                “number”: “303-555-1212”
            },
            “address”: {
                “premise”: “home”,
                “line1”: “555 Broadway NW”,
                “line2”: “# 000”,
                “city”: “Denver”,
                “stateOrProvince”: “CO”,
                “postalCode”: “88888”,
                “country”: “USA”
            }
        }
    ]
}
```

JSON Schema Generator

Creating a JSON Schema is tedious and error-prone. Use a JSON Schema Generator to generate a Schema from any valid JSON document. Visit the online JSON Schema Generator (www.jsonschema.net/) and generate a schema by doing the following:

- Paste the JSON document into the right-hand text area.
- Choose the JSON Input option
- Press the Generate Schema button.

JSON Schema Validator

An application uses a JSON Schema Validator to ensure that a JSON document conforms to the structure specified by the Schema. JSON Schema Validators are available for most modern programming languages:

JSON Schema Validator	Language	Source
JSV	JavaScript	https://github.com/garycourt/JSV
Ruby JSON Schema Validator	Ruby	https://github.com/hoxworth/json-schema

JSON Schema Validator	Language	Source
json-schema-validator	Java	https://github.com/fge/json-schema-validator
php-json-schema (by MIT)	PHP	https://github.com/hasbridge/php-json-schema
JSON.Net	.NET	http://james.newtonking.com/projects/json-net.aspx

Besides the language-specific tools, there is an excellent online JSON Schema Validator at: http://json-schema-validator.herokuapp.com. To use this site, enter the JSON document and Schema into the corresponding text boxes and press the `Validate` button.

IN CONCLUSION

We've covered the basics of JSON, but we've just scratched the surface. Although JSON is a simple data format, there are many tools to streamline the design and development process. JSON is a standard, it has replaced XML as the preferred data interchange format on the Internet, and it enables developers to create efficient, interoperable enterprise-class applications.

ABOUT THE AUTHORS

Tom Marrs is a Principal Consultant/Architect at Ciber, where he specializes in Service-Oriented Architecture (SOA) He designs and implements mission-critical web and business applications using the latest SOA, Ruby on Rails, REST, HTML5, JavaScript, Java/EE, and Open Source technologies. Tom is also the author of the upcoming JSON at Work, and a founder of the Denver Open Source User Group (DOSUG)

ISBN-13: 978-1-936502-71-4
ISBN-10: 1-936502-71-2
50795
9 781936 502714
$7.95

Version 1.0