# JSON Overview

# Copyright

## Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0)

This is a human-readable summary of (and not a substitute for) the license.

See: **https://creativecommons.org/licenses/by-nc-nd/4.0/** for the full details.

You are free to:
**Share** — copy and redistribute the material in any medium or format
The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:
**Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

**NonCommercial** — You may not use the material for commercial purposes.

**NoDerivatives** — If you remix, transform, or build upon the material, you may not distribute the modified material.

**No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

**Notices:**
You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.


Jonathan Earl
Jonathan@Earl-Family.Net

# Table of Contents


Copyright....2
Introduction....5
Background of JSON....6
- Origins....6
- Current JSON Standards....6
- Lab: JSON Standards....7
  - Exploring the Standards....7

Where JSON is used....8
- Web Applications....8
- NoSQL Databases....8
- Configuration Files....9
- HTML 5 Local Storage....10
- Node.js....11
- Lab: JSON Usage....11
  - Using JSON in a REST style Web Service....11
  - Using JSON as a configuration format....12

JSON Syntax....13
- JSON Objects....13
- JSON Arrays....14
- JSON Fields....14
- Lab: JSON Syntax....15
  - Working with JSON Objects....15
  - Working with JSON Arrays....15
  - Working with JSON Fields....15
  - Recognizing JSON Errors....15
- Lab: Working with Complex JSON....16
  - Initial Setup....16
  - JSON Objects within JSON Objects....16
  - JSON Arrays....16
  - Built-in JSON Data Types....17

JSON vs XML....18
- Lab: JSON vs XML....20
  - XML Style Web Service....20
  - JSON Style Web Service....20
  - Which is Better?....20

JSON Style Guide....21
- Lab: JSON Style....22
  - JSON Reserved Words....22
  - JSON Error Style....22
  - JSON Paging Style....22

JSON Schema....23
- Lab: JSON Schema....24



Jonathan Earl
Jonathan@Earl-Family.Net

Reading the Documentation....24
Creating a JSON Schema....24
Reading an Existing JSON Schema....24
JSON Security....25
JSON Injection....25
Preventing Server-Side JSON Injection....26
Preventing Client-Side JSON Injection....26
Lab: JSON Security....27
JSON Standards at the NSA....27
JSON Standards at the IRS....27
Coding JSON....28
JavaScript....28
Converting JSON to JavaScript Objects....28
Converting JavaScript Object to JSON....28
Java....28
Converting JSON to Java Objects....28
Converting Java Objects to JSON....29
Python....29
Converting JSON to Python....29
Converting Python Objects to JSON....29
Best Practices....30
Planning....30
Schema....30
Security....30
Testing....31
Documentation....31
Versioning....31
Follow the Standards....31
References....32
Glossary....34


Jonathan Earl
Jonathan@Earl-Family.Net

# Introduction

This is an overview of the JSON data format includes: the background of JSON, comparing it to XML, where JSON is used in today's environment and other related subjects.  The intent of this document is to educate the reader in the rules and use of JSON, and expose the reader to JSON security and coding considerations.

This document contains hands-on exercise that are designed to reinforce the subject at hands and allow the reader to explore JSON further.

There are a number of URLs referenced in this document, these URLs are listed in the Reference section at the end of the document.

There is also a Glossary at the end of this document defining the acronyms found in this document.


Jonathan Earl
Jonathan@Earl-Family.Net

# Background of JSON

## Origins

To quote Wikipedia:

> JSON (JavaScript Object Notation) is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format, with a diverse range of applications, such as serving as a replacement for XML in AJAX systems.

In the early 2000s, the exchange of data between a web server and a browser was dependent on the platform or the coding language. At this time Java Applets or Flash applications were the prominent methods of exchanging data.

A language and platform neutral data exchange format was required. In 2001 Douglas Crockford at State Software created a new data exchange format based on the JavaScript language. This format was name JSON indicating its JavaScript heritage. This format was proposed to the ECMA, the standards organization that defines the JavaScript standard (ECMA Script as ECMA-262). In 2013 ECMA-404 was published as the definitive JSON standard.

Prior to the formal adaption of the JSON standard, the JSON.org website was launched in 2002, and Yahoo started offering the JSON format for some of its Web Services in 2005.

In 2013 just after ECMA published its JSON standard, the IETF published RFC-7159 based on ECMA-404.

Douglas Crockford added a clause to the JSON license stating "The Software shall be used for Good, not Evil" when open-sourcing the JSON libraries. This clause led to license compatibility issues with open-source software that was addressed in the current JSON standard.

There are a number of extensions to the JSON standard, including the Dojo Toolkit which defines object references in JSON and JSON-LD from the W3C which defines Linked Data formats.

## Current JSON Standards

The current JSON standards include (see References for URLs):

- ECMA-404
- RFC 8259
- ISO/IEC 21778:2017


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: JSON Standards

## Exploring the Standards

Please open a Web Browser to:

**https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf** and read the Introduction to the ECMA-404 standard.

**https://tools.ietf.org/html/rfc8259** and read the Introduction section of RFC 8259 from the IETF.


Jonathan Earl
Jonathan@Earl-Family.Net

# Where JSON is used

JSON started out as a web specific data format, this is currently the most common usage. Over time JSON usage has extended to include NoSQL databases, configuration files, and HTML5 local storage uses. JSON as a data serialization format is growing in popularity.

## Web Applications

JSON is the de facto standard for REST (Representational State Transfer) style web services, and is used extensively in Ajax Web Applications. REST style Web Services use URL components and verbs to define the requesed action and response, for example the following URL will perform a HTTP GET request for the list of world currency codes: **https://openexchangerates.org/api/currencies.json**. This HTTP GET URL **https://api.weather.gov/points/39.7456,-97.0892** will describe the geographic point at Latitude 39.7456 and Longitude -97.0892, this point being 7366.9851976444 meters from the center of Linn KS.

In addition to REST style Web Services, JSON has become very popular in responsive Web Applications, replacing XML as the data exchange format of choice. This has resulted in a new anachronism instead of AJAX: Asynchronous JavaScript and XML, we now refer to AJAJ: Asynchronous JavaScript and JSON.

Using the Blue Nile jewelry web site, and searching for diamonds via: **https://www.bluenile.com/diamond-search** will result in an AJAJ style responsive web application. The best way to see this interaction is to open the web site, then open the Browser Developer Tools to examine the network traffic that is a response to User actions on the page.

## NoSQL Databases

JSON is a popular standard for data exchanges used in NoSQL databases. To quote the MongoDB documentation:

> In MongoDB, data is stored as documents. These documents are stored in MongoDB in JSON (JavaScript Object Notation) format. JSON documents support embedded fields, so related data and lists of data can be stored with the document instead of an external table.

Apache Cassandra supports JSON in it’s query language:

```
INSERT INTO student_registration
    JSON '{
        "s_id" : "9001",
        "s_name" : "Ashish",
        "s_email": "a_json1@gmail.com"
    }';
```


Jonathan Earl
Jonathan@Earl-Family.Net

Apache CouchDB stores it's data as JSON style key:value pairs. From the CouchDB introduction:

> CouchDB is a database that completely embraces the web. Store your data with JSON documents. Access your documents with your web browser, via HTTP. Query, combine, and transform your documents with JavaScript.

Traditional Relational databases such as Oracle are starting to embrace JSON. From the Oracle documentation:

> JSON data has often been stored in NoSQL databases such as Oracle NoSQL Database and Oracle Berkeley DB. These allow for storage and retrieval of data that is not based on any schema, but they do not offer the rigorous consistency models of relational databases.
>
> To compensate for this shortcoming, a relational database is sometimes used in parallel with a NoSQL database. Applications using JSON data stored in the NoSQL database must then ensure data integrity themselves.
>
> Native support for JSON by Oracle Database obviates such workarounds. It provides all of the benefits of relational database features for use with JSON, including transactions, indexing, declarative querying, and views.

## Configuration Files

JSON is slowly replacing YAML as the configuration file format of choice.

To deploy an application to the Google Cloud you require a JSON configuration file. Here is an example:

```
{
  "deployment": {
    "files": {
      "example-resource-file1": {
        "sourceUrl": "https://storage.googleapis.com/[MY_BUCKET_ID]/example-
application/example-resource-file1"
      },
      "images/example-resource-file2": {
        "sourceUrl": "https://storage.googleapis.com/[MY_BUCKET_ID]/example-
application/images/example-resource-file2"
      },
    }
  },
  "id": "v1",
  "handlers": [
    {
      "urlRegex": "/.*",
      "script": {
        "scriptPath": "example-python-app.py"
```


Jonathan Earl
Jonathan@Earl-Family.Net

```
      }
    },
  ],
  "runtime": "python27",
  "threadsafe": true,
}
```

The W3C requests a JSON configuration file named w3c.json as part of the repository documentation for projects under the W3C organization umbrella. From the W3C documentation:

> Projects operating under the w3c organization (or related to W3C even if under other umbrellas) are encouraged to specify a w3c.json file at the root of their repository. The purpose of this file is to provide some metadata about repositories so that they can be processed automatically by a variety of tools layered atop the organization. They can also help humans figure out who to contact for a given problem.

Here is an example:

```
{
   "group":     40318,
   "contacts":  ["darobin", "sideshowbarker"],
   "repo-type": "rec-track"
}
```

The GeoJSON standard was created to address locations such as Points, LineString, Polygon etc. Here is a GeoJSON example for a Point:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

## HTML 5 Local Storage

HTML 5 uses JSON to store key:value data pairs in both Local Storage and Session Storage. Here is an example:

```
<script>
if(localStorage) {
    localStorage.setItem("first_name", "Peter");

    alert("Hi, " + localStorage.getItem("first_name"));
}
</script>
```


Jonathan Earl
Jonathan@Earl-Family.Net

# Node.js

Node.js has evolved into a very popular development and deployment platform. You will see references to the MEAN stack (MongoDB, ExpressJS, AngularJS, and Node.js).

Node.js is a JavaScript development environment, and has a number of add-on packages to support for example, Web Applications and Databases. Node.js is configured via a package.json file.

# Lab: JSON Usage

## Using JSON in a REST style Web Service.

Open a Browser to

**http://api.worldbank.org/v2/countries/BR?format=json** this will query the World Bank for Brazil.

Try querying for Canada (CA), Japan (JP) or another country.

Try setting the format to XML (or leaving the format parameter off).

Open this URL **http://api.worldbank.org/v2/country?region=ARB&format=json**

and see the list of countries in the Arab World (ARB).

Try changing the region to:

- EMU:Euro Area
- EUU:European Union
- WLD:World,
- EAP:East Asia & Pacific
- ECA:Europe & Central Asia
- LAC:Latin America & Caribbean,
- MNA:Middle East & North Africa
- SAS:South Asia
- SSA:Sub-Saharan Africa
- CSS:Caribbean small states,
- OSS:Other small states
- PSS:Pacific island small states
- SST:Small states


Jonathan Earl
Jonathan@Earl-Family.Net

## Using JSON as a configuration format.

Open a Browser to: **https://geojson.io/** and replace the current configuration (right side of the screen) with this configuration:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -0.1278,
          51.5074
        ]
      },
      "properties": {
        "prop0": "London UK"
      }
    }
  ]
}
```

This will select London UK.  Can you find Moscow RU?
(Hint Moscow is at: 55.7558° N, 37.6173° E)

What about Rome?


Jonathan Earl
Jonathan@Earl-Family.Net

# JSON Syntax

JSON has a very simple syntax. Most smart IDEs or text editors will display JSON syntax errors and color code JSON text files.

JSON like JavaScript is case sensitive.

A JSON file is typically named with a .json suffix.

JSON allows whitespace (space, linefeed, carriage return, tab) between elements for readability.

See: **https://www.json.org/json-en.html** for the complete JSON syntax.

# JSON Objects

JSON objects are delimited by a set of { }. The root element in a JSON will typically be a JSON object. For example the GeoJSON configuration file referred to earlier starts out with a JSON object:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -0.1278,
          51.5074
        ]
      },
      "properties": {
        "prop0": "London UK"
      }
    }
  ]
}
```

Within this file are embedded objects, as seen by the sets of matching { }.
This file has the following JSON Objects;

- The root Object
- Within features there are:
  - An unnamed Object containing
    - type
    - geometry is an Object containing:
      - type
      - coordinates
  - The properties Object containing:
    - prop0


Jonathan Earl
Jonathan@Earl-Family.Net

## JSON Arrays

An array in JSON is indicated by [ ] . The above GeoJSON example has an array named “features”, with two elements within it. Notice the commas delimiting the entries in the array. The first entry in the “features” array is an unnamed object, the second entry is the object named properties.

JSON arrays may contain any combinations of fields, objects, or arrays. There is no size constraint.

Using an array as the root JSON object is discouraged for security reasons which will be explored later in this document.

## JSON Fields

A field in JSON is simple a name followed by a colon ‘:’ and then the contents of that field. Field contents (also called value) may be:

- string ← double quotes are used
- number ← may contain an E/e for exponent
- array ← [ ] are used
- object ← { } are used
- true
- false
- null

Note JSON does not support a native JavaScript Date type, you should convert the date to a string.

JSON strings may include selected escape characters (using a backslash ‘\’):

- Backspace is replaced with \b
- Form feed is replaced with \f
- Newline is replaced with \n
- Carriage return is replaced with \r
- Tab is replaced with \t
- Double quote is replaced with \"
- Backslash is replaced with \\

Unicode characters are also supported in a JSON string (via a \u notation), for example:

{ “name” : “Hel\u008En” } would result in a name of **Helèn**, and

{ “address” : “42 \u0443\u043B\u0438\u0446\u0430” } would results in **42 улица** (street in Russian).


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: JSON Syntax

## Working with JSON Objects

Open a Browser to: **https://jsoneditoronline.org/** and create the following JSON data:

- A root object to hold your data
- Within the root object add a field called “name” which will contain another JSON object
- Add the following fields within the “name” object:
  - First Name
  - Last Name
- Within the root object add a field named “address” which will contain another JSON object
- Add the following fields within the “address” object:
  - Street
  - City
  - State
  - Postal Code

## Working with JSON Arrays

Modify the JSON data you are working with to add an array of phone numbers. Please add the following numbers:

- Home
- Cell
- Work

## Working with JSON Fields

Modify the JSON data you are working with to add the following fields within the root object:

- age ← a number, try integer and floating point
- canadian address ← put null into this field
- citizen ← put true/false into this field
- distance to the moon ← put 238,900 using exponents in this field

## Recognizing JSON Errors

Modify the JSON data you are working by removing a comma. What error is indicated? Try removing a { or [.


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: Working with Complex JSON

Open a Browser to **https://www.mockaroo.com/** a mock data generator.

## Initial Setup

On the Mockaroo web site make the following initial changes:

- Set the number of Rows to: 1
- Set the Format to JSON
- Make sure the Array option is selected
- Make sure the “include null values” is selected

Click the Preview Button to see the JSON data.

## JSON Objects within JSON Objects

On the Mockaroo web site make the following changes:

- Change the first_name Field Name to: name.first_name
- Change the last_name to: name.last_name
- Click the “Add another field button” and add the following fields
  - Field name: address.street – Type: Street Address (click the Type field to select this type)
  - address.city – City
  - address.state – State
  - address.zip – Postal Code

Click the Preview Button to see the JSON data.

## JSON Arrays

On the Mockaroo web site make the following changes:
Add the following fields (Field Name - Type):

- phone – JSON Array
- phone.cell – Phone
- phone.home – Phone
- phone.fax – Phone, set the blank to 50%

Click the Preview Button to see the JSON data.
You will see a collection on anonymous JSON Objects in the phone array.

Change the phone.xxx fields to phone.person.xxx fields.

Click the Preview Button to see the JSON data.
You will see a collection on person named JSON Objects in the phone array.


Jonathan Earl
Jonathan@Earl-Family.Net

## Built-in JSON Data Types

On the Mockaroo web site make the following changes:
Add the following fields (Field Name – Type):

- manager – Boolean
- comments – Blank

Click the Preview Button to see the JSON data.

Deselect the “Include null values checkbox”, and preview the data again. You will no longer see the comments field.


Jonathan Earl
Jonathan@Earl-Family.Net

# JSON vs XML

Both JSON and XML are data formats and each offers advantages and disadvantages. XML is the standard for SOAP style Web Services and JSON is the standard for REST style Web Services.

XML Advantages:

- XML Documents may be encrypted
- XML Documents may be digitally signed
- XML Documents follow a strict structure and may be transformed via XSLT
- XML Documents may be assembled from composite parts
- XML Documents are inter-operable between any number of environments
- XML Documents may contain other markup elements
- XML Documents may be validated against a DTD or a Schema
- XML supports extensive data types

XML Disadvantages:

- XML does not support arrays
- XML is more verbose than JSON
- XML requires a larger bandwidth for transmission
- XML uses open/close tag, adding redundancy to the document
- XML is more difficult to produce and consume

JSON Advantages:

- JSON is faster than XML
- JSON is more compact
- JSON supports null values
- JSON is easy to produce and consume
- JSON is easier to read

JSON Disadvantages:

- JSON does not provide error handling
- Mishandling un-trusted JSON may result in a security issue
- JSON does not have Namespaces
- Schema support on JSON is basic


Jonathan Earl
Jonathan@Earl-Family.Net

Favor XML over JSON when any of these are true:

- You need message validation
- You're using XSLT
- Your messages include a lot of marked-up text
- You need to interoperate with environments that don't support JSON
- You need to encrypt or sign the document

Favor JSON over XML when all of these are true:

- Messages don't need to be validated, or validating their deserialization is simple
- You're not transforming messages, or transforming their deserialization is simple
- Your messages are mostly data, not marked-up text
- The messaging endpoints have good JSON tools

When all the conditions are equal, favor JSON for two reasons:

- JSON is a lot lighter to parse than XML (CPU friendly)
- JSON requires lot less data to be transferred (Network friendly)


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: JSON vs XML

## XML Style Web Service

Open a Browser to the National Weather Service XML (SOAP) Style Web Service:
**https://graphical.weather.gov/xml/SOAP_server/ndfdXMLclient.php?whichClient=NDFDgen&lat=40.015&lon=-105.2705&product=glance**

This is the weather forecast for Boulder CO, latitude 40.015 and longitude -105.270.

What is the expected low temperature tonight?

## JSON Style Web Service

Open a Browser to the National Weather Service JSON (REST) Style Web Service:
**https://api.weather.gov/gridpoints/BOU/53,74/forecast**

What is the expected low temperature tonight?

## Which is Better?

Which of the weather forecasts is better for a human to read?
Which has more data?
Which do you prefer? Why?
Which supports more complex data exchange?


Jonathan Earl
Jonathan@Earl-Family.Net

# JSON Style Guide

JSON is a flexible data format and whitespace is optional so both of the following examples are valid JSON:

```
{"id":1,"name":{"first_name":"Gradeigh","last_name":"Coultous"}
```

```
{
  "id": 1,
  "name": {
    "first_name": "Gradeigh",
    "last_name": "Coultous"
 }
```

The second example is much easier to read and modify.  Styling JSON allows humans to manage the the data, and avoid simple mistakes and misunderstandings.

Google publishes a JSON style guide that is the de facto standard: **https://google.github.io/styleguide/jsoncstyleguide.xml**

Here are a few of the highlights of this style guide:

- No comments in JSON
- Use double quotes
- Use camelCase for field names
- Use whitespace for readability
- Choose meaningful names
- Array names should be plural
- Field names should be singular
- Dates should be formatted as RFC 3339
- Time durations should be formatted as ISO 8601
- Latitudes/Longitudes should be formatted as ISO 6709

The style guide offers suggestion for JSON paging, errors, and links.


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: JSON Style

## JSON Reserved Words

Open a Browser to the Google JSON Style Guild

(**https:google.github.io/styleguide/jsoncstyleguide.xml**)

Find the Reserved Word list (Appendix A).

Notice the JSON reserved words are derived from the JavaScript (ECMAScript) reserved word list.

## JSON Error Style

Within the Google JSON Style Guild, find the recommended style for error objects.

## JSON Paging Style

Open a Browser to the World Bank REST Web Service:

**http://api.worldbank.org/country?per_page=10&region=WLD&format=json**

Does the World Bank follow the Google JSON Style Guide for paging?

Add the following to the end of the World Bank URL above: &page=2

Do the results show the correct page number?

What happens when you request a page the does not exist? page=25

What happens when you change the region to region=WLX (an error)?


Jonathan Earl
Jonathan@Earl-Family.Net

# JSON Schema

To quote json-schema.org:

> JSON Schema is a vocabulary that allows you to annotate and validate JSON documents.

Defining a schema for JSON data provides:

- A description for the data
- Documentation for the data
- Validates the data for testing
- Ensures data quality

Here is an example of a JSON schema:

```
{
   "$schema": "http://json-schema.org/2019-09/schema#",
   "title": "Product",
   "description": "A product from Acme's catalog",
   "type": "object",

   "properties": {

      "id": {
         "description": "The unique identifier for a product",
         "type": "integer"
      },

      "name": {
         "description": "Name of the product",
         "type": "string"
      },

      "price": {
         "type": "number",
         "minimum": 0,
         "exclusiveMinimum": true
      }
   },

   "required": ["id", "name", "price"]
}
```

There are a number of existing JSON Schema, see **https://www.schemastore.org/json/** for a repository of JSON Schemas.


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: JSON Schema

## Reading the Documentation

Open a Browser to: **https://www.ietf.org/archive/id/draft-bhutton-json-schema-00.txt** and read Section 3 Overview.

Please note there are now two media types to support JSON:

- application/json → for JSON documents
- application/schema+json → for JSON Schema documents

## Creating a JSON Schema

Open a Browser to **https://www.mockaroo.com/** and using the defaults to generate a simple 1 row JSON sample. Click Preview button and copy the JSON data.

Open a new Browser tab to **https://extendsclass.com/json-schema-validator.html** and paste the Mackaroo data into the left side window.

Click the Generate Schema From JSON button.

Change the id field from an integer to a string by putting double quotes around the value for the id field.

Notice the error message, when the data no longer validates against the schema.

What happens if you make the id field a floating point number? Does it matter if the value is 1.0 or 1.5?

## Reading an Existing JSON Schema

Open a Browser to **https://www.schemastore.org/json/**, and explore the following schemas:

- CityJSON – The JSON Schema for 3D City models.
- package.json – The JSON Schema for Node NPM packages


Jonathan Earl
Jonathan@Earl-Family.Net

# JSON Security

JSON by itself is not a security issue, JSON is a data exchange format. JSON by itself is just a document. Security concerns occur in the way JSON is used.

JSON used as a configuration format or as NoSQL document storage is not an issue, when JSON is used in a Web Application the door is opened to mishandling of the data.

Converting JSON to native JavaScript could introduce an number of web vulnerabilities. The ability to inject arbitrary JavaScript as part of a data transfer opens the door to a Cross Site Script (XSS) attack or a Cross Site Request Forgery (XSRF) attack. These are among the OWASP Top Ten web attacks, see: **https://owasp.org/www-project-top-ten/**.

The following is a simple JSON injection attack:

1. The web page builds a JSON object using data from the HTML form.

```
var result = eval("(" + json_string + ")");
document.getElementById("#account").innerText = result.account;
document.getElementById("#user").innerText = result.name;
document.getElementById("#pass").innerText = result.pass;
```

2. The Hacker enters this into the account field of the form:

```
user"});<script>document.location(“http://MySpoofingSite.com”);</
script>({"account":"user
```

3. Resulting in JavaScript in the JSON document.
4. As result of the JavaScript eval() function, the web page is relocated to the spoofing site and the User has no clue.

A Web Application consists of the Server-Side handling and the Client-Side handling. It is on the Client-Side that JSON security concerns appear. Client-Side usually means HTML, JavaScript and CSS.

## JSON Injection

The term “JSON injection” may be used to describe two primary types of security issues:

- Server-side “JSON injection” happens when data from an untrusted source is not sanitized by the server and written directly to a JSON stream.
- Client-side “JSON injection” happens when data from an untrusted JSON source is not sanitized and parsed directly using the JavaScript eval function.


Jonathan Earl
Jonathan@Earl-Family.Net

## Preventing Server-Side JSON Injection

It’s important to use a formal JSON parser and sanitizer when handling untrusted JSON on the server side. For example, the Java Programming language can utilize the OWASP JSON Sanitizer for Java.

A JSON sanitizer would take this code:

```
[ “<img src=”http://www.hacker.com/steal_data” />” ]
```

And turn it into this code:

```
[ “&lt;img src=”http://www.hacker.com/steal_data” /&gt;” ]
```

Notice the < and >, these will display ‘<’ and ‘>’, but run as JavaScript.

If you are doing JSON 100% properly, then you will only have objects at the top level. Arrays, Strings, Numbers, etc will all be wrapped. A JSON object will then fail to eval() because the JavaScript interpreter will think it's looking at a block rather than an object.

## Preventing Client-Side JSON Injection

Preventing JSON Injection Client-Side, is wholly dependent on the JavaScript command used. The eval() function should not be used, it will execute the injected JavaScript found in the JSON data.

JavaScript has a parse() function that is much safer and will not result in executable JavaScript. It should be the function of choice when building JavaScript object from a JSON document.

It is also recommended to have a JSON Object as the root element in your JSON data, avoid using a JSON Array as the root element. A JSON Object would not evaluate as embedded JavaScript, whereas a JSON Array would.

Apply exception handlers in the appropriate place as JSON.parse() can throw an exception.

Don't make assumptions about what data is there, you must explicitly test for data before using it.

Only process properties you are specifically looking for (avoiding other things that might be in the JSON).

Validate all incoming data as legitimate, acceptable values.

Sanitize the length of data (to prevent DOS issues with overly large data).

Don't put this incoming data into places where it could be further evaluated such as directly into the HTML of the page or injected directly into SQL statements without further sanitization to make sure it is safe for that environment.


Jonathan Earl
Jonathan@Earl-Family.Net

# Lab: JSON Security

## JSON Standards at the NSA

Open a Browser to:
**https://apps.nsa.gov/iaarchive/library/reports/security_guidance_for_json.cfm**

Click the button to **Get File**. This is the Security Guidance for the Use of JavaScript Object Notation (JSON) and JSON Schema from the NSA.

Read Section 3 - JSON Recommendations.

## JSON Standards at the IRS

Open a Browser to:
**https://www.irs.gov/irm/part2/irm_02-005-003**

And search the IRS Programming and Source Code Standards for JSON references.


Jonathan Earl
Jonathan@Earl-Family.Net

# Coding JSON

## JavaScript

### Converting JSON to JavaScript Objects

```
var text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

var obj = JSON.parse(text);

<p id="demo"></p>

<script>
  document.getElementById("demo").innerHTML =
  obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

Notice the use of JSON.parse(), it is a Best Practice to avoid using JSON.eval() as this could execute embedded JavaScript in the JSON data and is a security issue.

### Converting JavaScript Object to JSON

```
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

The JSON.stringify() function converts JavaScript object to JSON objects.

## Java

JSON support was introduced in Java EE 7, prior to this release you were required to add a 3rd party framework to work with JSON in Java. The following example use the json.jar from json.org.

### Converting JSON to Java Objects

```
Object obj=JSONValue.parse(json_data);
JSONObject jsonObject = (JSONObject) obj;

String name = (String) jsonObject.get("name");
double salary = (Double) jsonObject.get("salary");
long age = (Long) jsonObject.get("age");
```


Jonathan Earl
Jonathan@Earl-Family.Net

### Converting Java Objects to JSON

```
JSONObject obj = new JSONObject();
obj.put("name", "foo");
obj.put("num", new Integer(100));
JSONArray codes = new JSONArray();
codes.add(“CO”);
codes.add(“NY”);
codes.add(“UT”);
obj.put(“codes”, codes);
obj.put("balance", new Double(1000.21));
obj.put("is_vip", new Boolean(true));
```

## Python

The Python Standard Library has built-in support for JSON.  As following code examples illustrates.

### Converting JSON to Python

```
json_string = """
{
    "researcher": {
        "name": "Ford Prefect",
        "species": "Betelgeusian",
        "relatives": [
            {
                "name": "Zaphod Beeblebrox",
                "species": "Betelgeusian"
            }
        ]
    }
}
"""
data = json.loads(json_string)
```

The data object is now a Python Dictionary. The json.loads() function converts a JSON string into a Python object.

### Converting Python Objects to JSON

```
json_string = json.dumps(data)
```

The json.dump(data, file_name) or json.dumps(data) function coverts Python object to JSON, json.dumps() builds a string, and json.dump() outputs to a file.


Jonathan Earl
Jonathan@Earl-Family.Net

# Best Practices

When producing or consuming JSON, there are a number of Best Practices to follow.

## Planning

Always plan before you code, you need to able to answer the following questions before you start writing your JSON application:

- What is the root JSON object
- How many JSON fields are required
- What is the type and name of each field
- What are the business rules for each field
- What error conditions and messages are possible

## Schema

When designing a JSON data exchange, documenting the data format is critical.  Defining and using a JSON schema will help define the data, and control “data creep” that can occur when the data exchange is ad-hock.

When choosing a JSON schema, the first step is to look for an industry standard you can use such as the W3C JSON schemhttps://github.com/owasp/json-sanitizera for Financial products **https://www.w3.org/community/fibo/wiki/FinancialProduct_example**, or a JSON schema for Quantum Chemistry **https://molssi-qc-schema.readthedocs.io/en/latest/**.

If there is no industry standard JSON schema, you should create a schema for your data and document it.  The JSON Schema web site **https://json-schema.org/** provides a template for you to reuse.

## Security

Always use a JavaScript JSON parse() function, do not use JSON eval() function.

Always use a HTTP POST to serve JSON, never a GET, this will prevent most CSRF attacks.

Always wrap Server-Side JSON output in a JSON root Object. Never wrap the JSON in a root Array, as an Array is valid JavaScript and be be executed, where an Object is not valid JavaScript and will not be executed.


Jonathan Earl
Jonathan@Earl-Family.Net

Always sanitize any JSON being sent from a server.

Always check the length of the JSON data, extremely long lengths are an indicator of injected data.

Validate the JSON against a JSON Schema if there is one available.

## Testing

Test your Server-Side JSON producer, include in the tests injected JSON, null values, schema violations and all business data values. Automate your tests so they can be re-executed as needed.

Test your Client-Side JSON consumer, include in the tests injected JSON, null values schema violations and all business data values. Automate your tests so they can be re-executed as needed.

## Documentation

Always document your JSON, consider creating a schema and reference JSON examples.

Document any JSON error string and include the cause of the error,

## Versioning

Adding a version number to your schema and JSON documents will make it easier to release new editions of your data.

## Follow the Standards

Do not “stretch” JSON beyond what it was designed to do.  Do not get “creative” and try to make the JSON format handle operations, it only transports data.

Use the correct HTTP verb to query, insert, update or delete on a REST Web Application.

Keep informed of the latest JSON standards, including the new JSON Streaming standard being proposed by the W3C.


Jonathan Earl
Jonathan@Earl-Family.Net

# References

The following URLs are referenced in this document:

**https://creativecommons.org/licenses/by-nc-nd/4.0/** - The Creative Commons license

**https://www.json.org/json-en.html** – The JSON web site

**https://en.wikipedia.org/wiki/JSON** – The JSON Wikipedia page

**https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf** – The ECMA-404 JSON standard

**https://tc39.es/ecma262/** - The ECMA Script (JavaScript) draft 2021 standard

**https://tools.ietf.org/html/rfc8259** – The current JSON standard

**https://tools.ietf.org/html/std90** – the IETF JSON Data Interchange Format standard

**https://www.iso.org/standard/71616.html** – The ISO/IEC JSON Standard

**https://dojotoolkit.org/reference-guide/1.7/dojox/json/ref.html#:~:text=Usage-,dojox.,%2C%20and%20cross%2https://github.com/owasp/json-sanitizerDsite%20referencing** – the Dojo JSON extension

**https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf** – the JavaScript standard

**https://www.w3.org/TR/json-ld11/** - The W3C JSON based Serialization for Linked Data standard

**https://w3c.github.io/w3c.json.html** – The W3C requirement for a JSON file in all W3C projects

**https://geojson.org/** - the GeoJson web site (IETF RFC 7946)

**https://geojson.io/** - On-line GeoJson map display tool

**https://www.w3.org/community/fibo/wiki/FinancialProduct_example** - The W3C JSON Schema example for Financial Products

**https://molssi-qc-schema.readthedocs.io/en/latest/** - The JSON Schema for Quantum Chemistry

**https://json-schema.org/** - The JSON Schema web site

**https://openexchangerates.org/api/currencies.json** – REST style Web Service to list World currency codes

**https://api.weather.gov/points/39.7456,-97.0892** – REST style Web Service to document a Latitude and Longitude point


Jonathan Earl
Jonathan@Earl-Family.Net

**https://www.bluenile.com/diamond-search** – AJAJ style Web Application

**https://docs.mongodb.com/guides/server/introduction/** - The MongoDB documentation

**https://docs.couchdb.org/en/stable/intro/** - The CouchDb documentation

**https://docs.oracle.com/database/121/ADXDB/json.htm/** - Using JSON in Oracle

**https://cloud.google.com/appengine/docs/admin-api/creating-config-files/** - Use JSON to deploy an application to the Google Cloud

**https://w3c.github.io/w3c.json.html** – The W3C JSON configuration file documentation

**https://www.w3schools.com/whatis/whatis_json.asp/** - The W3Schools JSON tutorial

**https://www.freeformatter.com/json-formatter.html/** - On-line JSON formatter

**https://www.mockaroo.com/** - A mock data generator.

**https://google.github.io/styleguide/jsoncstyleguide.xml** – the Google JSON Style Guide

**https://tools.ietf.org/html/rfc3339/** - Formatting Dates in JSON

**https://www.iso.org/iso-8601-date-and-time-format.html/** - Formatting Time Durations in JSON

**https://www.iso.org/standard/39242.html/** - Formatting Latitudes/Longitudes in JSON

**https://www.ietf.org/archive/id/draft-bhutton-json-schema-00.txt/** - The JSON Schema IETF standard

**https://extendsclass.com/json-schema-validator.html/** - On-line JSON Schema builder

**https://www.schemastore.org/json/** - A list of existing JSON Schemas

**https://www.cityjson.org/** - JSON files to define 3D city models

**https://www.npmjs.com/package/node/** - Node Package Manager

**https://github.com/owasp/json-sanitizer**/ - The OWASP JSON Sanitizer for Java

**https://owasp.org/www-project-top-ten/** - The OWASP Top Tens web attacks

**https://apps.nsa.gov/iaarchive/library/reports/security_guidance_for_json.cfm** – The NSA JSON Security Guidelines

**https://www.irs.gov/irm/part2/irm_02-005-003** – The IRS Programming and Source Code Standards


Jonathan Earl
Jonathan@Earl-Family.Net

# Glossary

AJAJ – Asynchronous JavaScript and JSON

AJAX – Asynchronous JavaScript and XML

DTD - Document Type Definition

ECMA – European Computer Manufacturers Association

IDE – Integrated Development Environment

IEC – International Electrotechnical Commission committee of the ISO

IETF - Internet Engineering Task Force

ISO – International Organization for Standardization

JSON – JavaScript Object Notation

MEAN – MongoDB, ExpressJS, AngularJS, and Node.js

NPM – Node Package Manager

NSA – National Security Agency

OWASP – Open Web Application Security Project

SOAP - Simple Object Access Protocol

W3C – World Wide Web Consortium

XML – Extensible Markup Language

XSTL - Extensible Stylesheet Language Transformations

YAML – Yet Another Markup Language


Jonathan Earl
Jonathan@Earl-Family.Net