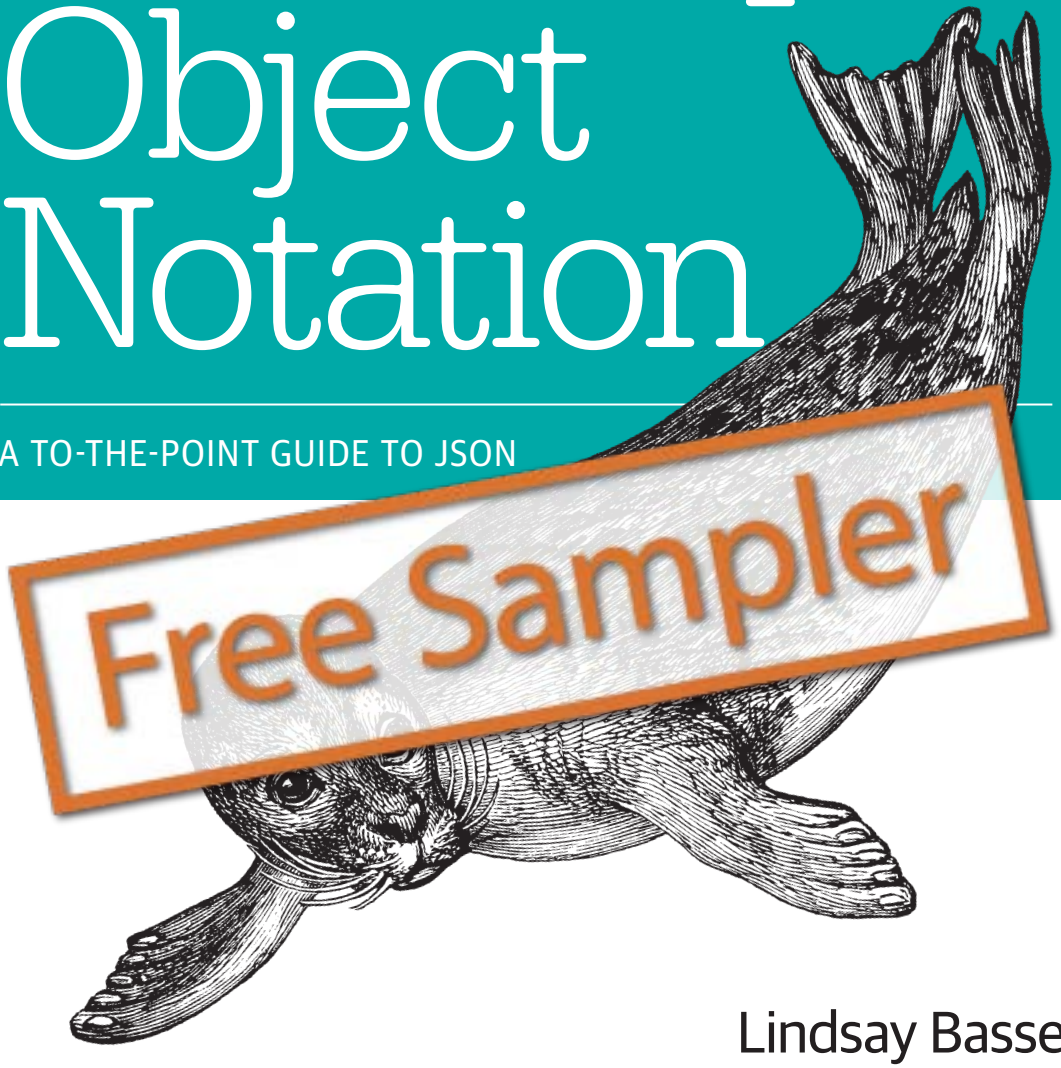
O'REILLY®

Introduction to JavaScript Object Notation

A TO-THE-POINT GUIDE TO JSON



Lindsay Bassett

O'REILLY®

Introduction to JavaScript Object Notation

What is JavaScript Object Notation (JSON) and how can you put it to work? This concise guide helps busy IT professionals get up and running quickly with this popular data interchange format, and provides a deep understanding of how JSON works. Author Lindsay Bassett begins with an overview of JSON syntax, data types, formatting, and security concerns before exploring the many ways you can apply JSON today.

From Web APIs and server-side language libraries to NoSQL databases and client-side frameworks, JSON has emerged as a viable alternative to XML for exchanging data between different platforms. If you have some programming experience and a basic understanding of HTML and JavaScript, this is your book.

- Learn why JSON syntax represents data in name-value pairs
- Explore JSON data types, including object, string, number, and array
- Find out how you can combat common security concerns
- Learn how the JSON schema verifies that data is formatted correctly
- Examine the relationship between browsers, web APIs, and JSON
- Understand how web servers can both request and create data
- Discover how jQuery and other client-side frameworks use JSON
- Learn why the CouchDB NoSQL database uses JSON to store data

Lindsay Bassett is an author, educator, and web developer with a passion for writing, teaching, and technology. Her online technology courses and books share the same "to-the-point" style that caters to IT professionals and students.

"A whole book, just on JSON? Yes. This book covers everything you need to know about JSON, and then goes beyond that into everything you need to know about JSON that you didn't know you needed. An excellent, informative, and exhaustive resource."

—Shelley Powers
Web Developer, Author of
JavaScript Cookbook and *HTML5 Media*

WEB PROGRAMMING

US $19.99 CAN $22.99
ISBN: 978-1-491-92948-3

Twitter: @oreillymedia
facebook.com/oreilly

Introduction to JavaScript Object Notation
by Lindsay Bassett

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editor: Meg Foley
Production Editor: Kristen Brown
Copyeditor: Jasmine Kwityn
Proofreader: Charles Roumeliotis
Indexer: Ellen Troutman
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

August 2015: First Edition

Revision History for the First Edition
2015-08-04: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491929483* for release details.

978-1-491-92948-3

[LSI]

Table of Contents

CHAPTER 1

What Is JSON?

Before we look at JSON from a low-level point of view, let's take a look at JSON from about 6,000 feet. From the mountain summit, we can see JSON flitting about in the world, carrying data in its lightweight format. If we look through our binoculars at JSON, we will see data among many curly bracket characters (`{}`). However, if we step back, and watch *how* it's being used, we will ultimately see that it is a data interchange format.

JSON Is a Data Interchange Format

A data interchange format is a text format used to exchange data between platforms. Another data interchange format you may already have heard of is XML. The world needs data interchange formats, like XML and JSON, to exchange data between very different systems.

Imagine for a moment a world comprised of hundreds of tiny, isolated islands among a vast ocean. Each island has its own unique language and customs. The islands all have seafaring merchants that travel long distances between the islands. Outside trade is an integral part of all the island economies and contributes to a high standard of living for the islanders. If it weren't for the highly trained carrier seagulls, this would not be possible.

The carrier seagulls move from island to island, carrying a paper report of data on which goods are in the highest demand. This way, merchants find out where they should move to next, and which goods they should acquire before embarking on their long voyages across the oceans. This important data allows all the islands to prosper without the threat of shortages.

Keep in mind, each island speaks a different language. If the data were passed around in several languages, each island would need to invest in researchers to learn all the

world's languages and employ a team of translators. This would be expensive and time consuming. This is an intelligent world, however, so the islands all agreed on a single language with a standard format for communicating their trade data. Each island employs just a single translator that understands the one data format of the trade reports brought by the carrier seagulls.

The real world of technology is much like the imagined island world example. There is a vast ocean, full of islands that have different languages, customs, and architecture. The ability for these unique systems to communicate is integral to many businesses and organizations. If each of these systems needed a translator for all the many ways other systems structure their data, then communications would consume an unreasonable amount of time and resources. Instead, the systems agree on a single format for data and employ a single translator.

JSON is a data interchange format that many systems have agreed on using for communicating data. You may hear it referred to as a "data exchange format," or simply a "data format." In this book, I will refer to JSON as a data interchange format because the definition of "interchange" reminds us that the data format is intended for two or more entities exchanging it.

Many, but not all systems have agreed on JSON for communicating data. There are data interchange formats, such as Extensible Markup Language (XML), that were around before JSON was even thought about. The real world is not quite as simple as the island example. Many systems have and still use other formats, such as XML, or more tabular, delimited formats such as comma-separated values (CSV). The decision by each island in the real world for which data format to accept for communication often has to do with how the data format relates to the customs, language, and architecture of the island.

In the island world example, each of the hundreds of islands had its own language. The data in the paper report that the carrier seagulls carried was in an agreed upon format that was independent of language. This way, a single translator of the trade reports data could be employed by each island. The same is true of JSON, except the data is carried across networks in zeros and ones instead of by seagulls. The translator isn't a human, it is a parser employed by the system consuming the data so it can be read within the system it is entering.

JSON Is Programming Language Independent

JSON stands for JavaScript Object Notation. The name of this data interchange format may mislead people into thinking they will need to learn JavaScript to understand and use JSON. There would be some value in learning JavaScript before learning JSON, as it was born out of a subset of JavaScript, but if you will not be using JavaScript anytime soon it would be unnecessary. You may remain dedicated to the

language or languages of your own island, for the spirit of a data interchange format is to be independent of language.

JSON is based on JavaScript object literals. A detailed explanation into the "how" of this is better suited for our discussion on syntax (Chapter 2) and data types (Chapter 3). For this chapter, the "why" is important. If a data interchange format is meant to be language independent, then it may seem contradictory to have a data format that is not only derived from a single language, but advertises it in its name: JavaScript Object Notation. Why, then?

If we return to the island example, imagine for a moment what the meeting to select the data format would have been like. When the representatives from each of the hundreds of islands arrived at this meeting, and looked to create a single data format, the first thing they would want to find is common ground.

The languages of each island may have been unique, but there were things the islanders found they had in common. Most of the languages were spoken primarily with the human voice and included a written form of the language represented by characters of some sort. Additionally, facial expressions and hand movements were also present. There were a few troublesome islands where the people communicated by other means, such as hitting sticks together or winking, but the majority of the islands found common ground with their written and spoken forms of language.

In the real world, there are hundreds of programming languages. Some are more popular and commonly used than others, but the language landscape is diverse. When college students major in computer science in preparation for a career in programming, they do not study all the programming languages. Students usually begin with one language, and the language itself is not so important as learning the universally accepted programming concepts. Once students gain an understanding of these concepts, they can more easily learn other programming languages through their ability to recognize the common features and functionalities.

If we set aside the word "JavaScript" from the name "JavaScript Object Notation," we would be left with "Object Notation." In fact, let's forget JavaScript all together. We could then say we are using an object notation data interchange format. "Object" is a common programming concept, in particular to object-oriented programming (OOP). Most computer science students studying programming will learn the concept of objects.

Without diving into an explanation of objects, let's settle our attention on the word "Notation." *Notation* implies a system of characters for representing data such as numbers or words. With or without an understanding of objects in programming, it is not a stretch to see the value of having a notation to describe something that is common across programming languages.

Returning again to the island example, the islanders themselves found a notation that represented a common tie among the majority of languages. Most of the islanders had a similar way of representing numbers with tallies, and it was agreed they could understand a series of symbols for representing real-world objects such as wheat or fabric. Even the island that communicated by winking found this format acceptable.

Despite the agreement among the vast majority of islands, there were still a few islands, such as the island that communicated by hitting sticks together, that did not find the format understandable. A good data interchange format covers the majority, but there are usually outliers. When we talk about this coverage, a term often thrown around is *portability*. Portability, or the compatibility in transferring information between platforms and systems, is the very goal of a data interchange format.

Circling back to notation, the notation of JSON may originate from JavaScript, but the notation itself is the important part. Not only is JSON language independent, it represents data in a way that speaks to common elements of many programming languages. With the way that data is represented, such as numbers and words, even the programming languages that aren't object oriented can find this format acceptable.

Key Terms and Concepts

This chapter covered the following key terms:

JSON
: JavaScript Object Notation.

Notation
: A system of characters for representing data such as numbers or words.

Data interchange format
: Text used to exchange data between platforms or systems.

Portability
: Transferring information between platforms in a way that is compatible with both systems.

We also discussed these key concepts:

- JSON is a data interchange format.
- JSON is programming language independent (JavaScript is not required to use it).
- JSON is based on the object literal notation of JavaScript (emphasis on the word "notation").
- JSON represents data in a way that is friendly to universal programming concepts.

CHAPTER 2

JSON Syntax

JSON Is Based on JavaScript Object Literals

In the English language, the word "literal" is an adjective used to imply that what is being said is exact, not a metaphor. When your friend says, "She showed up out of nowhere and I literally dropped my sandwich," he is stating that the dropping of the sandwich is not a metaphor.

In programming, the word "literal" is a noun. A *literal* is a value that is represented literally with data. It is written precisely as it is meant to be interpreted. If you aren't familiar with programming concepts, then this might seem strange. Let's take a quick look at literals.

Do you carry cash in your wallet, or a debit card? When I stop off at the sandwich shop and hand the cashier a five dollar bill for my sandwich, I physically watch my five dollars leave my wallet. When I swipe my debit card to pay for a sandwich, I know I have five dollars less in my back account, even though I didn't see it happen.

In programming, we often use variables to represent values. For example, I might use a *variable* I call *x* in an expression like:

```
x = 5
```

Then, later on, I might want to add five more to *x*:

```
x = x + 5
```

At this point, we know the value of *x* is 10, but we don't *see* 10. In this example, *x* was the variable, and 5 was a literal. In our sandwich shop example, we could say that the five dollars cash was a literal, and the debit card was a variable. When we *see* the actual value, it is a literal value.

In the "$x = 5$" example, 5 is a number literal. A number is a data type. Other types of data are strings (made up of characters), boolean (true or false), null (nothing), collections of values, and objects. Representing a number value in a way that we can see is simple, and we use a number character. Representing a boolean value is also simple and we can use `true/false` or `0/1`. If you are familiar with the concept of objects, you will understand that representing an object is no easy or simple matter. If you aren't familiar with the concept of objects, that is OK too.

In programming, the concept of an object is similar to how you would describe a real-world object, such as your shoes. You could describe your shoes with attributes or properties such as color, style, brand, and the type of insole. Some of these attribute values could be a number, such as shoe size, and others could be a boolean (true/false) such as "has laces." Example 2-1 shows an example.

Example 2-1. Using JSON to describe the shoes I'm wearing right now

```
{
    "brand": "Crocs",
    "color": "pink",
    "size": 9,
    "hasLaces": false
}
```

Don't be concerned just yet about the syntax in the shoe example; we will arrive there later in this chapter. The main point of the shoe example is that you (even a human) can literally read the attributes of my shoe. The data type for my JSON shoe example is object. The literal value of the object exposes the properties or attributes in a way which we can see (and read). These attributes or properties of the shoe object are represented as name-value pairs.

JSON is based on JavaScript object literals. The key phrase here is "based on." In JavaScript (and most programming languages with objects), the object can include a function. So not only could I represent the properties of my shoe with the JavaScript object, but I could create a function called "walk."

However, data interchange is about data, so JSON does not include the functions of JavaScript object literals. The way that JSON is based on JavaScript object literals is purely in the syntactic representation of the object literal and its properties. This representation of properties is achieved with name-value pairs.

Name-Value Pairs

The concept of name-value pairs is widespread in computing. They are called by other names as well: key-value pairs, attribute-value pairs, and field-value pairs. In this book, we will refer to them as name-value pairs.

If you are familiar with the concept of name-value pairs, JSON will seem natural to you. If you aren't familiar with name-value pairs, that's OK too. Let's take a quick look at name-value pairs.

In a name-value pair, you first declare the name. For example, `"animal"`. Now, pair implies two things: a name and a value. So let's give our name (in this case, `"animal"`) a value. To simplify this concept for this chapter, let's use a string value. With name-value pairs in JSON, the value can also be a number, a boolean, null, an array, or an object. We will go more in depth with the other value data types beyond string in Chapter 3. So, for this name-value pair, which has the name `"animal"`, we will use the string value, `"cat"`:

```
"animal" : "cat"
```

`"animal"` is the name and `"cat"` is the value. There are many ways that we could choose to delimit, or separate, the name and the value. If I were to provide you with a directory of a company's employees with their job titles, I'd probably hand you a list that looks something like this:

- Bob Barker, Chief Executive Officer
- Janet Jackson, Chief Operations Officer
- Mr. Ed, Chief Financial Officer

For my employee directory, I used commas to separate my job titles (names) and employee names (values). I also placed the value on the left and the name on the right.

JSON uses the colon character (`:`) to separate the names and values. The name is always on the left and the value is always on the right. Let's take a look at a few more:

```
"animal" : "horse"

"animal" : "dog"
```

Simple, right? A name and a value, and you have a name-value pair.

Proper JSON Syntax

Now let's take a look at what proper JSON syntax entails. The name, which in our example is `"animal"`, is always surrounded in double quotes. The name in the double quotes can be any valid string. So, you could have a name that looks like this, and it would be perfectly valid JSON:

```
"My animal": "cat"
```

You can even place an apostrophe in the name:

```
"Lindsay's animal": "cat"
```

Now that you know that this is valid JSON, I'm going to tell you why you shouldn't do this. The name-value pairs used in JSON are a friendly data structure to many systems. Having a space or special character (other than a–z, 0–9) in the name would not be taking *portability* into consideration. In Chapter 1, we defined this key term as "transferring information between platforms in a way that is compatible with both systems." We can do things in our JSON data that decrease portability; therefore, we say it is important to avoid spaces or special characters for *maximum portability*.

The name in the name-value pair of your JSON, if it is to be loaded in memory by a system as an object, will become a "property" or "attribute." A property or attribute in some systems can include an underscore character (_) or numbers, but in most cases it is considered good form to stick to the characters of the alphabet, A–Z or a–z. So, if I wanted to include multiple words in my name, I would format like so:

```
"lindsaysAnimal": "cat"
```

or

```
"myAnimal": "cat"
```

The `"cat"` value in the example has double quotes. Unlike the name in the name-value pair, the value does not always have double quotes. If our value is a string data type, we must have double quotes. In JSON, the remaining data types are number, boolean, array, object, and null. These will not be surrounded in double quotes. The format of these will be covered in Chapter 3.

JSON stands for JavaScript Object Notation. So, the only thing we are missing is the syntax that makes it an object. We need curly brackets surrounding our name-value pair to make it an object. So, one before...

```
{ "animal" : "cat" }
```

...and one after. When you are formatting your JSON, picture a knighting ceremony where the master of the ceremony dubs the new knight on the shoulders with a sword. You are the master of the ceremony, and you must dub your JSON as an object on each side with a curly bracket. "I dub thee, sir JSON." The ceremony would not be complete without a tap on each shoulder.

In JSON, multiple name-value pairs are separated by a comma. So, to extend the animal/cat example, let's add a color:

```
{ "animal" : "cat", "color" : "orange" }
```

Another way to look at JSON syntax would be through the eyes of the machine that is reading it. Unlike humans, machines are very rigidly rule- and instruction-oriented creatures. When you use any of the following characters outside of a string value (not surrounded in quotes), you are providing an instruction on how your data is to be read:

- { (left curly bracket) says "begin object"
- } (right curly bracket) says "end object"
- [(left square bracket) says "begin array"
-] (right square bracket) says "end array"
- : (colon) says "separating a name and a value in a name-value pair"
- , (comma) says "separating a name-value pair in an object" or "separating a value in an array"; can also be read as "here comes another one"

If you forget to say "end object" with a right curly bracket, then your object will not be recognized as an object. If you place a comma at the end of your list of name-value pairs, you are giving the instruction "here comes another one" and then not providing it. Therefore, it is important to be correct in your syntax.

A Story: The Double Quotes of JSON

One day I was peering over a student's shoulder, looking at his computer screen. He was showing me some JSON that he was about to validate (Example 2-2).

Example 2-2. The "JSON" that would not validate

```
{
    title : "This is my title.",
    body : "This is the body."
}
```

Upon validation he received a parsing error and became frustrated. He said, "Look, there's nothing wrong with it!"

I pointed out to him that he was missing quotes around `"title"` and `"body"`. He said, "But I've seen JSON formatted both ways, with and without quotes around the names." "Ah," I said. "When you saw it without quotes around the names, that was not JSON. It was a JavaScript object."

This confusion is understandable. JSON is based on JavaScript object literals, so it looks much the same. A JavaScript object literal does not need quotes around the name of the name-value pair. In JSON, it is absolutely required.

Another point of confusion can be the usage of single quotes instead of double quotes. In JavaScript, an object may have single quotes for syntax instead of double quotes (see Example 2-3).

Example 2-3. This is not valid JSON

```
{
    'title': 'This is my title.',
    'body': 'This is the body.'
}
```

In JSON, only double quotes are used, and they are absolutely required around the name of the name-value pair (see Example 2-4).

Example 2-4. Valid JSON

```
{
    "title": "This is my title.",
    "body": "This is the body."
}
```

Syntax Validation

Unlike machines, as a human using a keyboard, creating an error is as simple as a missed keystroke. It's amazing, really, that we don't produce more errors than we do. Validating JSON is an important part of working with JSON.

Your integrated development environment (IDE) might have built-in validation for your JSON. If your IDE supports plug-ins and add-ons, you might find a validation tool there if it is not already integrated. If you don't use an IDE, or have no idea what I'm talking about, that's OK too.

There are many online tools for formatting and validating JSON. A quick jaunt on your search engine for "JSON validation" will give you several results. Here are a few worth mentioning:

JSON Formatter & Validator
: A formatting tool with options, and a beautiful UI that highlights errors. The processed JSON displays in a window that doubles as a tree/node style visualization tool and a window to copy/paste your formatted code from.

JSON Editor Online
: An all-in-one validation, formatting, and visualization tool for JSON. An error indicator is displayed on the line of the error. Upon validation, helpful parsing error information is displayed. The visualization tool displays your JSON in a tree/node format.

JSONLint
: A no-bells-and-whistles validation tool for JSON. Simply copy, paste, and click "validate." It also kindly formats your JSON.

These are tools for *syntax validation*. Later, in Chapter 4, we'll discuss another type of validation called conformity validation. Syntax validation concerns the form of JSON itself, whereas conformity validation concerns a unique data structure. For Example 2-5, syntax validation would be concerned that our JSON is correct (surrounded in curly brackets, dividing our name-value pairs with commas). Conformity validation would be concerned that our data included a name, breed, and age. Addi-

tionally the conformity validation would be concerned that the value of age is a number, and the value of name is a string.

Example 2-5. Validation example

```
{
    "name": "Fluffy",
    "breed": "Siamese",
    "age": 2
}
```

JSON as a Document

You might find that in your future experiences with JSON, you are only ever creating it in code and passing it around in an unseen world that can only be inspected by developer tools. However, as a data interchange format, JSON can be its own document and live in a filesystem. The file extension for JSON is easy to remember: *.json*.

So, if I were to save my animal/cat JSON to a file and store it on my computer, it would look something like this: *C:\animals.json*.

The JSON MediaType

Oftentimes when you are passing data to someone else, you need to tell them ahead of time what type it is. You may hear this called an Internet media type, a content type, or a MIME type. This type is formatted as *`type/subtype`*. One type that you may have already heard of is `text/html`.

The MIME type for JSON is `application/json`.

The Internet Assigned Numbers Authority (IANA) maintains a comprehensive list of media types.

Key Terms and Concepts

This chapter covered the following key terms:

Literal
: A value that is written precisely as it is meant to be interpreted.

Variable
: A value that can be changed and is represented by an identifier, such as *x*.

Maximum portability (in data interchange)
: Transcending the base portability of the data format by ensuring the data itself will be compatible across systems or platforms.

Name-value pair
: A name-value pair (or key-value pair) is a property or attribute with a name, and a corresponding value.

Syntax validation
: Validation concerned with the form of JSON.

Conformity validation
: Validation concerned with the unique data structure.

We also discussed these key concepts:

- JSON is based on the syntactic representation of the properties of JavaScript object literals. This *does not* include the functions of JavaScript object literals.
- In the JSON name-value pair, the name is always surrounded by double quotes.
- In the JSON name-value pair, the value can be a string, number, boolean, null, object, or array.
- The list of name-value pairs in JSON is surrounded by curly brackets.
- In JSON, multiple name-value pairs are separated by a comma.
- JSON files use the *.json* extension.
- The JSON media type is `application/json`.