JSON

JavaScript Object Notation (JSON) is a data format that was built to be easily readable for both humans and computers. It has become the most popular way for applications to exchange data, and its most commonly encountered when working with APIs like the Nylas Email, Calendar, and Contacts connectivity APIs, but you'll also find JSON files stored on servers, used in data exports, and many other places.

The minimal syntax and simple parsing of JSON increases the speed of communication in multiple ways. It's often less data to download compared to other formats, and it can be interpreted in most modern languages without external libraries. For these reasons, along with being self-describing and easy-to-read, nearly 20 years after it was first created, JSON remains the data format choice for most developers.

JSON rose to popularity alongside browser programming languages that required compact and convenient serialization of data. It was originally intended to be a lightweight alternative to XML but has largely replaced it for easy parsing on the web. This guide provides historical context, useful benefits, and practical applications to better understand this important format.

What is JSON?

JSON is a data interchange format that is easy to parse and generate. JSON is an extension of the syntax used to describe object data in JavaScript. Yet, it's not restricted to use with JavaScript. It has a text format that uses object and array structures for the portable representation of data. All modern programming languages support these data structures, making JSON completely language independent.

JSON History

In the early 2000s Douglas Crockford created JSON to be minimal, portable, and textual. A subset of JavaScript (hence, the name), JSON came into popularity around the same time as the web browser scripting language. It was used widely at Yahoo!, where Crockford later worked as an architect. By the early 2010s, JSON was the popular choice for new public APIs.

It was standardized in 2013, as ECMA-404, and published in 2017 as RFC8259, the Internet Engineering Task Force (IETF) standard for the Internet. The RFC and ECMA standards remain consistent. The official Internet media type (also known as a Multipurpose Internet Mail Extensions or MIME type) for JSON is application/json, and JSON file names use the extension .json.

JSON grew out of a need for stateless, real-time server-to-browser communication protocol sans browser plugins like Flash or Java applets, the dominant methods used in the early 2000s. It aimed to be a lightweight alternative to XML to allow for mobile processing scenarios and easy parsing of JavaScript on the web.

JSON is commonly associated with REST services, especially for APIs on the web. Although the REST architecture for APIs allows for any format, JSON provides a more flexible message format that increases the speed of communication. It is useful when developing web or mobile applications where fast, compact, and convenient serialization of data is required.

JSON Structure, Syntax, and Usage

JSON's simplicity is part of its appeal. It's easy to write, easy to read, and easy to translate between the data structures used by most languages. Let's look at what makes up a JSON object, the data types that JSON supports, and other specifics with the syntax of this popular data format.

Chances are you've seen JSON as you've looked through data or API documentation. The criteria for valid JSON is rather elementary, though it can

be used to describe complex data. The structure of a JSON object is as follows:

- Curly braces {} hold objects
- The data are in key, value pairs
- Square brackets [] hold arrays
- Each data element is enclosed with quotes if it‘s a character, or without quotes if it is a numeric value
- Commas are used to separate pieces of data

Here’s a basic example:

```
{ "name":"Katherine Johnson" }
```

The key is “name” and the value is “Katherine Johnson” in the above example. However, JSON can hold more than one key:value pair. This second example adds an “age” key, which includes a number and a second string value, assigned to the “city” key:

```
{ "name":"Katherine Johnson", "age":101, "city":"Newport News" }
```

It’s common to encounter nested JSON structures, like this:

```
{"inventors":[

   { "name":"Katherine Johnson", "age":101, "city":"Newport News" },
```

```
    { "name":"Dorothy Vaughan", "age":98, "city":"Hampton" },

    { "name":"Henry Ford", "age":83, "city":"Detroit" }

]}
```

In this final example, you see a primary object with a single key (“inventors”) that has an array as its value. Within that array, each item is itself an object, similar to the earlier simple example. Objects and arrays are values that can hold other values, so there’s an unlimited nesting that could happen with JSON data. That allows JSON to describe most data types, from tabular to even more complex.

JSON Data Types

Now that you’ve seen the structure of JSON, you’ve been introduced to several of its data types. There are only a couple others to introduce. Here is the complete list of JSON data types:

- string – Literal text that’s enclosed in quotes.
- number – Positive or negative integers or floating point numbers.
- object – A key, value pair enclosed in curly braces
- array – A collection of one or more JSON objects.
- boolean – A value of either true or false with no quotes.
- null – Indicates the absence of data for a key value pair, represented as “null” with no quotes.

Here’s an example of a JSON object that includes all of these data types:

```
{

   "name":"Katherine Johnson",

   "age":101,

   "orbital_mechanics": ["trajectories","launch
windows","emergency return paths"],

   "mathmatician": true,

   "last_location": null

}
```

JSON Syntax

We've already discussed the structure of JSON, which provides the basics of the syntax. In this section, we'll suggest some best practices to avoid common JSON errors:

Always enclose the key, value pair within double quotes. Most JSON parsers don't like to parse JSON objects with single quotes.

```
{ "name": "Katherine Johnson" }
```

Never use hyphens in your key fields. Use underscores (_), all lower case, or camel case.

```
{ "first_name":"Katherine", "last_name":"Johnson" }
```

Use a JSON linter to confirm valid JSON. Install a command line linter or use an online tool like JSONLint. If you copy this next example into a JSON linter, you should get a parse error for the pesky single quotes around the value for last_name.

```
{ "first_name":"Katherine", "last_name":'Johnson' }
```

What Are the Benefits of JSON?

The rise in JSON's popularity coincides with a need for websites and mobile apps to more easily and efficiently transfer data from one system to another. But there are many ways that JSON is used to share data, store settings, and interact with systems. Its simplicity and flexibility make it applicable to a number of different situations.

While you'll find JSON files on servers around the web, the most common usage is to exchange serialized data over a network connection. Some other common uses for JSON include public, front-end, or internal APIs, NoSQL databases, schema descriptions, configuration files, public data, or data exports.

The benefits of JSON include:

- Compact, efficient format: JSON syntax offers easy parsing of data and even faster implementation
- Easily readable: Both humans and computers can quickly interpret the syntax with minimal errors
- Broadly supported: Most languages, operating systems, and browsers can consume JSON out of the box, which allows JSON to be used without compatibility concerns
- Self-describing: It's easy to distinguish between data types and makes it easier to interpret the data without knowing what to expect ahead of time
- Flexible format: JSON supports a wide range of data types that can be combined to express the structure of most data

How Does JSON Compare to the Alternatives?

There are many data formats for developers to choose from including YAML, XML, and CSV. In this section, we'll take a look at these alternatives and how they compare to JSON.

Extensible Markup Language (XML) vs. JSON

Before there was JSON, there was XML. The format is still widely used, but has fallen out of favor versus the efficient and flexible JSON. Nevertheless, you will still find ample support for XML, especially within large enterprises and those using toolsets with robust XML support.

The primary argument against XML is that it's inefficient, redundant, and overly repetitive, sort of like this sentence. XML uses tags (similar to HTML) to deliver data. Consider this JSON example with three inventors we showed earlier. The equivalent in XML would read:

```
<?xml version="1.0" encoding="UTF-8" ?>

<inventors>

   <inventor>

       <name>Katherine Johnson</name>

       <age>101</age>

       <city>Newport News</city>

   </inventor>

   <inventor>

       <name>Dorothy Vaughan</name>

       <age>98</age>

       <city>Hampton</city>

   </inventor>

   <inventor>

       <name>Henry Ford</name>
```

```
        <age>83</age>

        <city>Detroit</city>

    </inventor>

</inventors>
```

Notice that each field (name, age, city) is listed twice. The array or list of inventors must be explicitly created and each element noted again (<inventors> and three <inventor> tags). As you can see, XML is verbose when compared to the simplicity of JSON syntax. After minimizing whitespace, the above examples weigh in at 177 characters for JSON and 310 for XML. That's 77% more data to express the same information.

Despite these drawbacks, XML has a rich history. It was standardized before the wide usage of APIs and even has web service standards built on top of it, notably the SOAP messaging protocol. XML is stricter than JSON and has support for schemas and namespaces, which is crucial when passing data between separate systems.

While XML parsers are built-in into all modern browsers, cross-browser XML parsing can be tricky. Also, parsing large XML files can be a large performance hit. For web pages, JSON is easier to load, read and manipulate.

YAML Ain’t Markup Language (YAML) vs. JSON

YAML is a superset of JSON that’s primarily designed to support more complex requirements by allowing user-defined data types as well as explicit data typing. YAML is typically used to represent configuration information in a simple key, value format using whitespace to show structure. The extensions for YAML files are .yaml or .yml; here’s a YAML version of the JSON example we saw earlier:

```yaml
inventors:

- name: Katherine Johnson

 age: 101

 city: Newport News

- name: Dorothy Vaughan

 age: 98

 city: Hampton

- name: Henry Ford

 age: 83

 city: Detroit
```

Note the usage of whitespace, which makes it easy to read. While JSON can be minimized and maintain its structure, YAML uses whitespace to create the structure, so it loses meaning when removed.

JSON is better as a serialization format for serving data because it is more explicit and suitable for data interchange between APIs. However, YAML has its advantages. To start, it supports comments, which allow developers and other users to provide human-readable (and machine-ignorable) context to data. The single-line comments show YAML's focus on humans first, and YAML's extreme simplicity, plus usage of whitespace, means it's generally easier to read.

There are some more advanced features of YAML too. For example, it can store recursive data structures with the *anchor* node property that's represented with an ampersand. These anchors, allow you to avoid repetition and can be added to any mapping, sequence or scalar. Later in the YAML Document you can refer to a previously defined anchor with an alias, which indicates that the node with that anchor name should occur in both places.

Finally, YAML allows for embedding other serialization formats, such as JSON or XML within a YAML file. So, while YAML is a competitor to JSON, you might find yourself using both.

Comma-Separated Values (CSV)

The final data format we'll cover is a classic: Comma-Separated Values (CSV) files have been used since the earliest days of computers. In fact, CSV may even pre-date computers. This simple format can be stored in flat files and easily passed between systems. However, it is not particularly robust in the type of data it can express.

You can think of CSV like a simple Excel spreadsheet. There are rows and columns, with each cell containing one piece of data. In fact, the first line of a CSV is often the headers. And if compact data is what you're after, CSV has you covered. Here's a comparable CSV version to the examples used in the previous sections:

```
name,age,city

Katherine Johnson,101,Newport News

Dorothy Vaughan,98,Hampton

Henry Ford,83,Detroit
```

It's a pretty simple representation of the data, but it's also pretty simplistic data. CSV cannot handle nested data without converting it into a single line representation, which generally makes it difficult to use. For simple data exports, such as records with a single level of hierarchy, CSV makes sense. As a data format for the complex needs of today's applications, it's a non-starter.

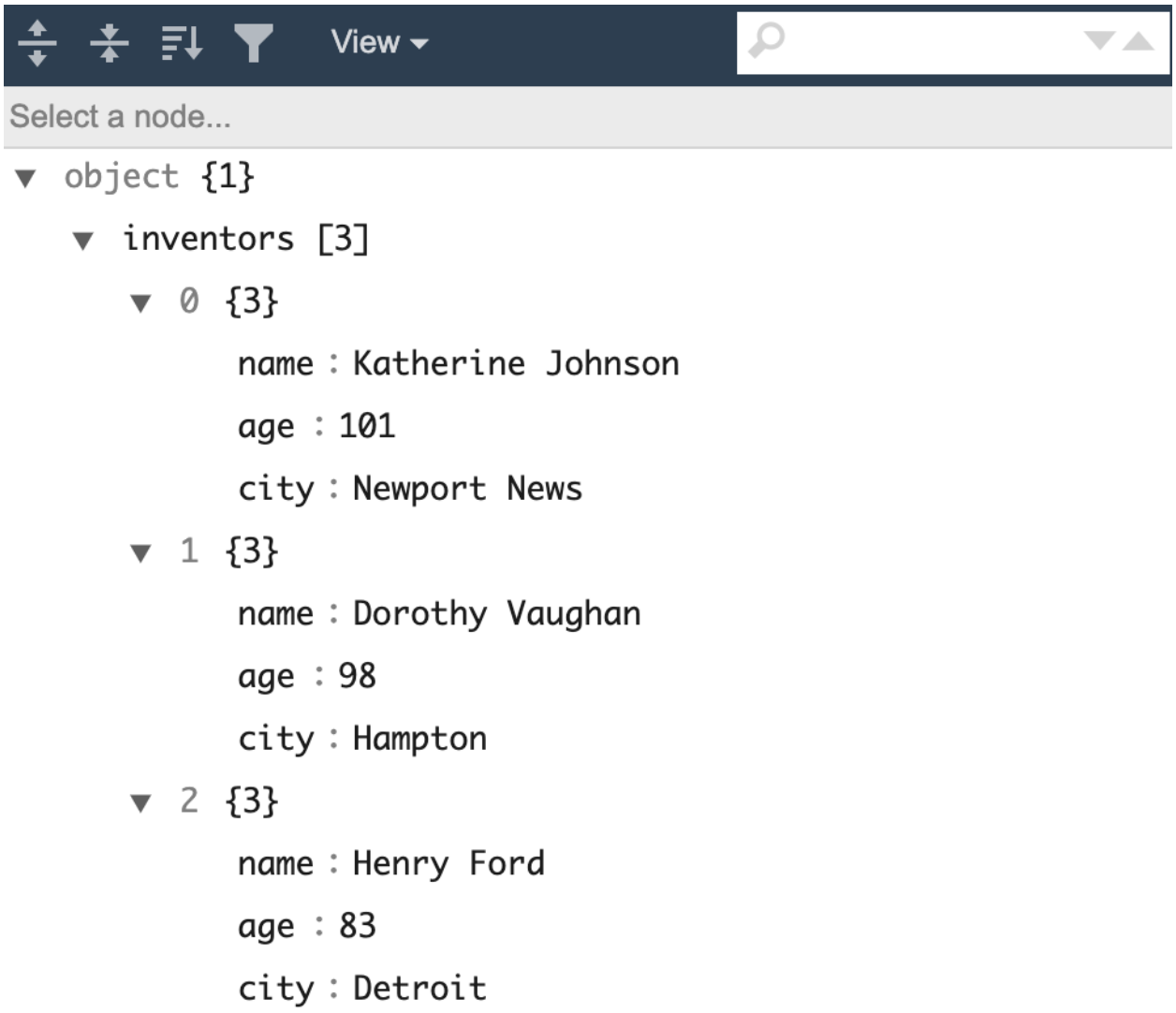
How to Use JSON

The first thing you need to use JSON is a parser that can interpret JSON data. The examples shown in this guide are primarily printed with indentation and newlines for clarity. When your applications receive JSON, it typically comes in one massive concatenation of characters. Inside of this string are encoded tags as well as key, value pairs. A JSON parser takes this huge string and breaks it up into the data structures that are indicated by the string.

An online JSON parser can help you visualize the structure of your data. For example, consider this minimized version of the inventors data we've used throughout this guide:

```
{"inventors":[{"name":"Katherine
Johnson","age":101,"city":"Newport News"},{"name":"Dorothy
Vaughan","age":98,"city":"Hampton"},{"name":"Henry
Ford","age":83,"city":"Detroit"}]}
```

The online JSON parser will display the data like so:



Most popular programming languages have built in support to parse JSON, but they also include a plethora of JSON manipulation functions including the ability to modify JSON, save to and read from JSON files, and convert common data objects into data formats. You can read our guide on using

Python to work with JSON to learn more, or check out the standard libraries for JavaScript, Java, Ruby, and C#.

JSON Makes the World Go Round

JSON has become a vital tool of the developer arsenal because of its ease of use, efficiency, and flexibility. It's no wonder that developers everywhere are choosing to use it. In fact, we use JSON here at Nylas for our Email, Calendar, and Contacts APIs. If you want to use JSON for your next communications and scheduling integration, sign up for a free trial today