http://www.w3schools.com/website/web_spa_json.asp

Web Building - Fetching Data with JSON

Building a web site from scratch. **Part VII**: Fetching data using JSON.

What We Will Do

In this chapter we will:

- Fetch data from a text file using JSON

The JSON File

The following file is stored on our web server:

http://www.w3schools.com/website/Customers_JSON.php

```
[
{
"Name" : "Alfreds Futterkiste",
"City" : "Berlin",
"Country" : "Germany"
},
{
"Name" : "Berglunds snabbköp",
"City" : "Luleå",
"Country" : "Sweden"
},
{
"Name" : "Centro comercial Moctezuma",
"City" : "México D.F.",
"Country" : "Mexico"
},
{
"Name" : "Ernst Handel",
"City" : "Graz",
"Country" : "Austria"
},
{
"Name" : "FISSA Fabrica Inter. Salchichas S.A.",
"City" : "Madrid",
"Country" : "Spain"
},
{
```

```
"Name" : "Galería del gastrónomo",
"City" : "Barcelona",
"Country" : "Spain"
},
{
"Name" : "Island Trading",
"City" : "Cowes",
"Country" : "UK"
},
{
"Name" : "Königlich Essen",
"City" : "Brandenburg",
"Country" : "Germany"
},
{
"Name" : "Laughing Bacchus Wine Cellars",
"City" : "Vancouver",
"Country" : "Canada"
},
{
"Name" : "Magazzini Alimentari Riuniti",
"City" : "Bergamo",
"Country" : "Italy"
},
{
"Name" : "North/South",
"City" : "London",
"Country" : "UK"
},
{
"Name" : "Paris spécialités",
"City" : "Paris",
"Country" : "France"
},
{
"Name" : "Rattlesnake Canyon Grocery",
"City" : "Albuquerque",
"Country" : "USA"
},
{
"Name" : "Simons bistro",
"City" : "København",
"Country" : "Denmark"
},
{
"Name" : "The Big Cheese",
"City" : "Portland",
"Country" : "USA"
},
{
"Name" : "Vaffeljernet",
"City" : "Århus",
"Country" : "Denmark"
},
{
"Name" : "Wolski Zajazd",
"City" : "Warszawa",
"Country" : "Poland"
}
]
```

Change the Customers Page to use JSON

In the **DemoWeb** folder, change the file **Customers.html**.

Put the following code inside the file:

Customers.html

```
<!DOCTYPE html>
<html>

<head>
  <title>Our Company</title>
  <link href="Site.css" rel="stylesheet">
</head>

<body>

<nav id="nav01"></nav>

<div id="main">
  <h1>Customers</h1>
  <div id="id01"></div>
  <footer id="foot01"></footer>
</div>

<script src="Script.js"></script>

<script>
var xmlhttp = new XMLHttpRequest();
var url = "http://www.w3schools.com/website/Customers_JSON.php";

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        myFunction(xmlhttp.responseText);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
</script>
```

```
<script>
function myFunction(jsonText) {
   var myList = JSON.parse(jsonText);
   var arr = myList.Customers;
   var i;
   var out = "<table>";
   for(i = 0; i < arr.length; i++) {
      out += "<tr><td>" +
      arr[i].Name +
      "</td><td>" +
      arr[i].City +
      "</td><td>" +
      arr[i].Country +
      "</td></tr>";
   }
   out += "</table>"
   document.getElementById("id01").innerHTML = out;
}
</script>

</body>
</html>
```

Try it Yourself »

Read More

Read more about JSON in our JSON Tutorial.

JSON Tutorial

JSON: **J**ava**S**cript **O**bject **N**otation.

JSON is a **syntax** for storing and exchanging data.

JSON is an **easier to use** alternative to XML.

This JSON syntax defines an employees object, with an array of 3 employee records (objects):

JSON Example

```
{"employees":[
   {"firstName":"John", "lastName":"Doe"},
   {"firstName":"Anna", "lastName":"Smith"},
   {"firstName":"Peter", "lastName":"Jones"}
]}
```

This XML syntax also defines an employees object with 3 employee records:

XML Example

```
<employees>
   <employee>
      <firstName>John</firstName> <lastName>Doe</lastName>
   </employee>
   <employee>
      <firstName>Anna</firstName> <lastName>Smith</lastName>
   </employee>
   <employee>
      <firstName>Peter</firstName> <lastName>Jones</lastName>
   </employee>
</employees>
```

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is lightweight data interchange format
- JSON is language independent *****
- JSON is "self-describing" and easy to understand

* JSON uses JavaScript syntax, but the JSON format is text only, just like XML.
Text can be read and used as a data format by any programming language.

JSON - Introduction

JSON - Evaluates to JavaScript Objects

The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, instead of using a parser (like XML does), a JavaScript program can use standard JavaScript functions to convert JSON data into native JavaScript objects.

Try it Yourself

With our editor, you can edit JavaScript code online and click on a button to view the result:

JSON Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON Object Creation in JavaScript</h2>

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

<script>
var text = '{"name":"John Johnson","street":"Oslo West 16","phone":"555 1234567"}'

var obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
obj.name + "<br>" +
obj.street + "<br>" +
obj.phone;
</script>

</body>
</html>
```

Try it yourself »

Much Like XML

- Both JSON and XML is plain text
- Both JSON and XML is "self-describing" (human readable)
- Both JSON and XML is hierarchical (values within values)
- Both JSON and XML can be fetched with an HttpRequest

Much Unlike XML

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

The biggest difference is:

XML has to be parsed with an XML parser, JSON can be parsed by a standard JavaScript function.

Why JSON?

For AJAX applications, JSON is faster and easier than XML:

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

JSON Syntax

The JSON syntax is a subset of the JavaScript syntax

JSON Syntax Rules

JSON syntax is a subset of the JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"firstName":"John"

JSON Values

JSON values can be:

- A number (integer or floating point)
- A string (in double quotes)
- A Boolean (true or false)
- An array (in square brackets)
- An object (in curly braces)
- null

JSON Objects

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/values pairs:

```
{"firstName":"John", "lastName":"Doe"}
```

JSON Arrays

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain multiple objects:

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
```

In the example above, the object "employees" is an array containing three objects. Each object is a record of a person (with a first name and a last name).

JSON Uses JavaScript Syntax

Because JSON uses JavaScript syntax, no extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an array of objects and assign data to it like this:

Example

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

The first entry in the JavaScript object array can be accessed like this:

```
employees[0].firstName + " " + employees[0].lastName;
```

The returned content will be:

```
John Doe
```

The data can be modified like this:

```
employees[0].firstName = "Gilbert";
```

Try it yourself »

In the next chapter you will learn how to convert a JSON text to a JavaScript object.

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

JSON How To

A common use of JSON is to read data from a web server, and display the data in a web page.

For simplicity, this can be demonstrated by using a string as input (instead of an file).

JSON Example - Object From String

Create a JavaScript string containing JSON syntax:

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

JSON syntax is a subset of JavaScript syntax.

The JavaScript function JSON.parse(*text*) can be used to convert a JSON text into a JavaScript object:

```
var obj = JSON.parse(text);
```

Use the new JavaScript object in your page:

Example

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

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

Try it yourself »

Using eval()

Older browsers without the support for the JavaScript function JSON.parse() can use the eval() function to convert a JSON text into a JavaScript object:

```
var obj = JSON.parse(text);
```

Try it yourself »

The eval() function can compile and execute any JavaScript.
This represents a potential security problem. **Try to avoid it**.

It is safer to use a JSON parser to convert a JSON text to a JavaScript object.

A JSON parser will recognize only JSON text and will not compile scripts.

In browsers that provide native JSON support, JSON parsers are also faster.

Native JSON support is included in all major browsers and in the latest ECMAScript (JavaScript) standard:

Web Browsers Support

- Firefox 3.5
- Internet Explorer 8
- Chrome
- Opera 10
- Safari 4

For older browsers, a JavaScript library is available at https://github.com/douglascrockford/JSON-js

The JSON format was originally specified by Douglas Crockford

JSON Files

A common use of JSON is to read data from a web server, and display the data in a web page.

This page will teach you how in 4, very easy to follow, steps.

JSON Example

This example reads a menu from W3Schools, and displays the menu in a web page:

```
<div id="w3schools"></div>

<script>
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i<arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
    document.getElementById("w3schools").innerHTML = out;
}
</script>

<script src="w3schools.com"></script>
```

Try it yourself »

Example Explained

1: Create an array of objects.

This is an array of objects. Each object has two properties: "display" and "url".

```
var myArray = [
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]
```

2: Create a JavaScript function to display the array.

This function loops the objects of the array, and display the content as HTML links:

```
<div id="w3schools"></div>

<script>
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i<arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
    document.getElementById("w3schools").innerHTML = out;
}
</script>
```

This statement calls myFunction with myArray as argument:

```
myFunction(myArray);
```

Try it yourself »

3: Use an array literal as the argument (instead of an array variable):

This statement calls myFunction with an array literal as argument:

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
```

```
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp""
}
]);
```

Try it yourself »

4: Put the function call in an external js file

Put this in an external js file. Name the file "w3schools.js"

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp""
}
]);
```

Add the external script to your page (instead of the function call):

```
<div id="w3schools"></div>

<script>
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i<arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
    document.getElementById("w3schools").innerHTML = out;
}
</script>

<script src="w3schools.com"></script>
```

Try it yourself »

Example:
http://www.w3schools.com/json/json_http.asp

```
<!DOCTYPE html>
<html>
<body>

<div id="id01"></div>

<script>
```

```
var xmlhttp = new XMLHttpRequest();
var url = "myTutorials.txt";

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var myArr = JSON.parse(xmlhttp.responseText);
        myFunction(myArr);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' +
        arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
</script>

</body>
</html>
```

myTutorials.txt

```
[
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp""
}
]
```

http://nitschinger.at/Handling-JSON-like-a-boss-in-PHP

06.06 | Handling JSON like a boss in PHP

After reading this post, you will be able to understand, work and test JSON code with PHP.

There are already lots of tutorials out there on handling JSON with PHP, but most of them don't go much deeper than throwing an array against json_encode and hoping for the best. This article aims to be a solid introduction into JSON and how to handle it correctly in combination with PHP. Also, readers who don't use PHP as their programming language can benefit from the first part that acts as a general overview on JSON.

JSON (JavaScript Object Notation) is a data exchange format that is both lightweight and human-readable (like XML, but without the bunch of markup around your actual payload). Its syntax is a subset of the JavaScript language that was standardized in 1999. If you want to find out more, visit the official website.

The cool thing about JSON is that you can handle it natively in JavaScript, so it acts as the perfect glue between server- and client-side application logic. Also, since the syntactical overhead (compared to XML) is very low, you need to transfer less bytes of ther wire. In modern web stacks, JSON has pretty much replaced XML as the de-factor payload format (aside from the Java world I suppose) and client side application frameworks like backbone.js make heavy use of it inside the model layer.

Before we start handling JSON in PHP, we need to take a short look at the JSON specification to understand how it is implemented and what's possible.

Introducing JSON

Since JSON is a subset of the JavaScript language, it shares all of its language constructs with its parent. In JSON, you can store unordered key/value combinations in objects or use arrays to preserve the order of values. This makes it easy to parse and to read, but also has some limitations. Since JSON only defines a small amount of data types, you can't transmit types like dates natively (you can, but you need to transform them into a string or a unix timestamp as an integer).

So, what datatypes does JSON support? It all boils down to strings, numbers, booleans and null. Of course, you can also supply objects or arrays as values.

Here's a example JSON document:

```
{
   "title": "A cool blog post",
   "clicks": 4000,
   "children": null,
   "published": true,
   "comments": [
      {
         "author": "Mister X",
         "message": "A really cool posting"
      },
      {
         "author": "Misrer Y",
         "message": "It's me again!"
      }
   ]
}
```

It contains basically everything that you can express through JSON. As you can see no dates, regexes or something like that. Also, you need to make sure that your whole JSON document is encoded in UTF-8. We'll see later how to ensure that in PHP. Due to this shortcomings (and for other good reasons) BSON(Binary JSON) was developed. It was designed to be more space-efficient and provides traversability and extensions like the date type. Its most prominent use case is MongoDB, but honestly I never came across it somewhere else. I recommend you to take a short look at the specification if you have some time left, since I find it very educating.

Since PHP has a richer type handling than JSON, you need to prepare yourself to write some code on both ends to transform the correct information apart from the obligatory encoding/decoding step. For example, if you want to transport date objects, you need to think if you can just send a unix timestamp over the wire or maybe use a preformatted date string (like strftime).

Encoding JSON in PHP

Some years ago, JSON support was provided through the json pecl extension. Since PHP 5.2, it is included in the core directly, so if you use a recent PHP version you should have no trouble using it.

Note: If you run an older version of PHP than 5.3, I recommend you to upgrade anyway. PHP 5.3 is the oldest version that is currently supported and with the latest PHP security bugs found I would consider it critical to upgrade as soon as possible.

Back to JSON. With json_encode, you can translate anything that is UTF-8 encoded (except resources) from PHP into a JSON string. As a rule of thumb, everything except pure arrays (in PHP this means arrays with an ordered, numerical index) is converted into an object with keys and values.

The method call is easy and looks like the following:

```
json_encode(mixed $value, int $options = 0);
```

An integer for options you might ask? Yup, that's called a bitmask. We'll cover them in a separate part a little bit later. Since these bitmask options change the way the data is encoded, for the following examples assume that we use defaults and don't provide custom params.

Let's start with the basic types first. Since its so easy to grasp, here's the code with short comments on what was translated:

```
<?php
// Returns: ["Apple","Banana","Pear"]
json_encode(array("Apple", "Banana", "Pear"));

// Returns: {"4":"four","8":"eight"}
json_encode(array(4 => "four", 8 => "eight"));

// Returns: {"apples":true,"bananas":null}
json_encode(array("apples" => true, "bananas" => null));
?>
```

How your arrays are translated depends on your indexes used. You can also see that `json_encode` takes care of the correct type conversion, so `booleans` and `null` are not transformed into strings but use their correct type. Let's now look into objects:

```
<?php
class User {
   public $firstname = "";
   public $lastname  = "";
   public $birthdate = "";
}

$user = new User();
$user->firstname = "foo";
$user->lastname  = "bar";

// Returns: {"firstname":"foo","lastname":"bar"}
json_encode($user);
```

```
$user->birthdate = new DateTime();

/* Returns:
    {
        "firstname":"foo",
        "lastname":"bar",
        "birthdate": {
            "date":"2012-06-06 08:46:58",
            "timezone_type":3,
            "timezone":"Europe\/Berlin"
        }
    }
*/
json_encode($user);
?>
```

Objects are inspected and their public attributes are converted. This happens recursively, so in the example above the public attributes of the DateTime object are also translated into JSON. This is a handy trick if you want to easly transmit datetimes over JSON, since the client-side can then operate on both the actual time and the timezone.

```
<?php
class User {
    public $pub = "Mister X.";
    protected $pro = "hidden";
    private $priv = "hidden too";

    public $func;
    public $notUsed;

    public function __construct() {
        $this->func = function() {
            return "Foo";
        };
    }
}

$user = new User();

// Returns: {"pub":"Mister X.","func":{},"notUsed":null}
echo json_encode($user);
?>
```

Here, you can see that only public attributes are used. Not initialized variables are translated to `null` while closures that are bound to a public attribute are encoded with an empty object (as of PHP 5.4, there is no option to prevent public closures to be translated).

The $option bitmasks

Bitmasks are used to set certain flags on or off in a function call. This language pattern is commonly used in C and since PHP is written in C this concept made it up to some PHP function arguments as well. It's easy to use: if you want to set an option, just pass the constant as an argument. If you want to combine two or more options, combine them with the bitwise OR operation `|`. So, a call to json_encode may look like this:

```
<?php
// Returns: {"0":"Starsky & Hutch","1":123456}
json_encode(array("Starsky & Hutch", "123456"), JSON_NUMERIC_CHECK |
JSON_FORCE_OBJECT);
?>
```

`JSON_FORCE_OBJECT` forces the array to be translated into an object and `JSON_NUMERIC_CHECK` converts string-formatted numbers to actual numbers. You can find all bitmasks (constants) here. Note that most of the constants are available since PHP 5.3 and some of them were added in 5.4. Most of them deal with how to convert characters like `< >`, `&` or `""`. PHP 5.4 provides a `JSON_PRETTY_PRINT` constant that may you help during development since it uses whitespace to format the output (since it adds character overhead, I won't enable it in production of course).

Decoding JSON in PHP

Decoding JSON is as simple as encoding it. PHP provides you a handy json_decode function that handles everything for you. If you just pass a valid JSON string into the method, you get an object of type `stdClass` back. Here's a short example:

```
<?php
$string = '{"foo": "bar", "cool": "attr"}';
$result = json_decode($string);

// Result: object(stdClass)#1 (2) { ["foo"]=> string(3) "bar" ["cool"]=>
string(4) "attr" }
var_dump($result);

// Prints "bar"
echo $result->foo;

// Prints "attr"
echo $result->cool;
?>
```

If you want to get an associative array back instead, set the second parameter to true:

```
<?php
$string = '{"foo": "bar", "cool": "attr"}';
$result = json_decode($string, true);

// Result: array(2) { ["foo"]=> string(3) "bar" ["cool"]=> string(4) "attr" }
var_dump($result);

// Prints "bar"
echo $result['foo'];

// Prints "attr"
echo $result['cool'];
?>
```

If you expect a very large nested JSON document, you can limit the recursion depth to a certain level. The function will return `null` and stops parsing if the document is deeper than the given depth.

```
<?php
$string = '{"foo": {"bar": {"cool": "value"}}}';
$result = json_decode($string, true, 2);
```

```
// Result: null
var_dump($result);
?>
```

The last argument works the same as in json_encode, but there is only one bitmask supported currently (which allows you to convert bigints to strings and is only available from PHP 5.4 upwards).We've been working with valid JSON strings until now (aside fromt the `null` depth error). The next part shows you how to deal with errors.

Error-Handling and Testing

If the JSON value could not be parsed or a nesting level deeper than the given (or default) depth is found, NULL is returned from json_decode. This means that no exception is raised by json_encode/json_deocde directly.

- So how can we identify the cause of the error? The json_last_error function helps here. json_last_error returns an integer error code that can be one of the following constants (taken from here):
- JSON_ERROR_NONE: No error has occurred.
- JSON_ERROR_DEPTH: The maximum stack depth has been exceeded.
- JSON_ERROR_STATE_MISMATCH: Invalid or malformed JSON.
- JSON_ERROR_CTRL_CHAR: Control character error, possibly incorrectly encoded.
- JSON_ERROR_SYNTAX: Syntax error.
- JSON_ERROR_UTF8: Malformed UTF-8 characters, possibly incorrectly encoded (since PHP 5.3.3).

With those information at hand, we can write a quick parsing helper method that raises a descriptive exception when an error is found.

```
<?php
class JsonHandler {

   protected static $_messages = array(
      JSON_ERROR_NONE => 'No error has occurred',
      JSON_ERROR_DEPTH => 'The maximum stack depth has been exceeded',
      JSON_ERROR_STATE_MISMATCH => 'Invalid or malformed JSON',
      JSON_ERROR_CTRL_CHAR => 'Control character error, possibly incorrectly
encoded',
      JSON_ERROR_SYNTAX => 'Syntax error',
      JSON_ERROR_UTF8 => 'Malformed UTF-8 characters, possibly incorrectly
encoded'
   );

   public static function encode($value, $options = 0) {
      $result = json_encode($value, $options);

      if($result)  {
         return $result;
      }

      throw new RuntimeException(static::$_messages[json_last_error()]);
   }

   public static function decode($json, $assoc = false) {
      $result = json_decode($json, $assoc);
```

```
        if($result) {
            return $result;
        }

        throw new RuntimeException(static::$_messages[json_last_error()]);
    }

}
?>
```

We can now use the exception testing function from the last post about exception handling to test if our exception works correctly.

```
// Returns "Correctly thrown"
assertException("Syntax error", function() {
    $string = '{"foo": {"bar": {"cool": NONUMBER}}}';
    $result = JsonHandler::decode($string);
});
```

Note that since PHP 5.3.3, there is a `JSON_ERROR_UTF8` error returned when an invalid UTF-8 character is found in the string. This is a strong indication that a different charset than UTF-8 is used. If the incoming string is not under your control, you can use the utf8_encode function to convert it into utf8.

```
<?php echo utf8_encode(json_encode($payload)); ?>
```

I've been using this in the past to convert data loaded from a legacy MSSQL database that didn't use UTF-8.

Summary

JSON is a convenient, readable and easy to use data exchange format that seems to replace XML as the de-facto standard on the web. PHP has everything you need already built in and provides various configuration options if you use a recent version (> 5.3).

You should now be able to understand JSON, how to interact with it through PHP and how to handle possible errors. Happy encoding!

(By the way, you may also want to check out my primer about Exceptions in PHP!)