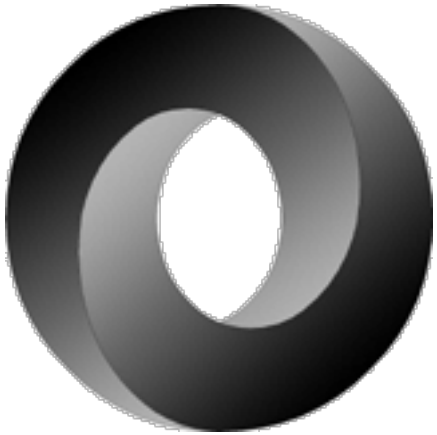
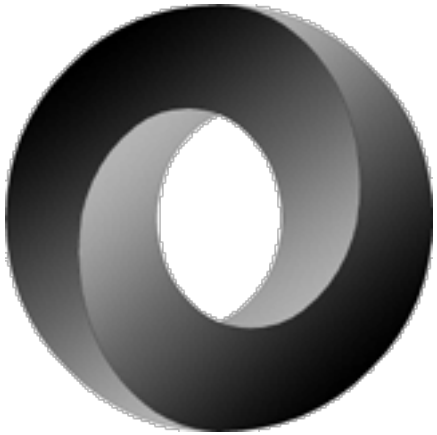


JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.



JSON is built on two structures:

A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

```
var tourDates = {concerts:
[
   {
      "id": 0,
       "tourloc":"Manchester, NH",
      "tourVenue": "Verizon Wireless Arena",
      "tourDate": "Sep-7"
   },
   {
      "id": 1,
       "tourloc":"Washington, DC",
      "tourVenue": "Jiffy Lube Live Amphitheater",
      "tourDate": "Sep-9"
   }
]};
```

JSONLint

The JSON Validator

A Tool from the Arc90 Lab. Source is on GitHub.
Props to Douglas Crockford of JSON and JS Lint and
Zach Carter, who provided the pure JS implementation of jsonlint.

```
1  Enter JSON to validate, or a URL to JSON to validate.
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Validate

JSON Lint is an idea sparked at Arc90 by

Kindling™ { INNOVATION THROUGH COLLABORATION }

An Idea Management & Collaboration Tool

FAQ

Templating

http://mustache.github.com/

Basic Template

```
var person = {
    firstName: "Christopher",
    lastName: "Griffith",
    blogURL: "http://chrisgriffith.wordpress.com"
};
var template = "<h1>{{firstName}} {{lastName}}</h1>Blog: {{blogURL}}";
var html = Mustache.to_html(template, person);
$('#sampleArea').html(html);
```

Christopher Griffith

Blog: http://chrisgriffith.wordpress.com

Basic Template using Ajax data

```
$.getJSON('js/data.json', function(data) {
        var template = "<h1>{{firstName}} {{lastName}}</h1>Blog: {{blogURL}}";
        var html = Mustache.to_html(template, data);
        $('#demo2').html(html);
    });
```

Christopher Griffith

Blog: http://chrisgriffith.wordpress.com

Externalized Template

```
var template = $('#personTpl').html();
var html = Mustache.to_html(template, data);
$('#sampleArea').html(html);
```

```
<script id="personTpl" type="text/template">
<h1>{{firstName}} {{lastName}}</h1>
<p>Blog: <a href="{{blogURL}}">{{blogURL}}</a></p>
</script>
```

Christopher Griffith

Blog: http://chrisgriffith.wordpress.com

Enumerable Section with Objects

```
var data = {
    employees: [
    {   firstName: "Chris",
        lastName: "Griffith"},
    {   firstName: "John",
        lastName: "Smith"}
    ]};
var template = "Employees:<ul>{{#employees}}" +
                        "<li>{{firstName}} {{lastName}}</li>" +
                        "{{/employees}}</ul>";
var html = Mustache.to_html(template, data);
$('#sampleArea').html(html);
```

Employees:

- Chris Griffith
- John Smith

Nested Objects

```
var person = {
    firstName: "Chris",
    lastName: "Griffith",
    blogURL: "http://chrisgriffith.wordpress.com",
    manager : {
        firstName: "Bob",
        lastName: "Ross"
    }
};
var template = "<h1>{{firstName}} {{lastName}}</h1><p>{{blogURL}}</p>" +
            "Manager: {{manager.firstName}} {{manager.lastName}}";
var html = Mustache.to_html(template, person);
$('#sampleArea').html(html);
```

Chris Griffith

http://chrisgriffith.wordpress.com

Manager: Bob Ross

Function

```
var product = {
    name: "FooBar",
    price: 100,
    salesTax: 0.05,
    totalPrice: function() {
        return this.price + this.price * this.salesTax;
    }
};
var template = "<p>Product Name: {{name}}</p>Price: {{totalPrice}}";
var html = Mustache.to_html(template, product);
$('#sampleArea').html(html);
```

Product Name: FooBar

Price: 105

Condition

Templates can include conditional sections. Conditional sections only render if the condition evaluates to true.
A conditional section begins with {{#condition}} and ends with {{/condition}}. “condition” can be a boolean value or a function returning a boolean.

Condition

```
var data = {
    employees: [
    {   firstName: "Christopher",
        lastName: "Griffith",
        fullTime: true,
        phone: "617-123-4567"
    },
    {   firstName: "John",
        lastName: "Smith",
        fullTime: false,
        phone: "617-987-6543"
    },
    {   firstName: "Bob",
        lastName: "Ross",
        fullTime: true,
        phone: "617-111-2323"
    },
    ]};
var tpl = "Employees:<ul>{{#employees}}<li>{{firstName}} {{lastName}}" +
          "{{#fullTime}} {{phone}}{{/fullTime}}</li>{{/employees}}</ul>";
var html = Mustache.to_html(tpl, data);
$('#sampleArea').html(html);
```

Condition

Employees:

- Christopher Griffith 617-123-4567
- John Smith
- Bob Ross 617-111-2323

Application Cache

Using the cache interface gives your application three advantages:

1. Offline browsing - users can navigate your full site when they're offline
2. Speed - cached resources are local, and therefore load faster.
3. Reduced server load - the browser will only download resources from the server that have changed.

The Cache Manifest File

```
<html manifest="example.appcache">
  ...
</html>
```

The manifest attribute should be included on every page of your web application that you want cached.

Structure of a Manifest File

```
CACHE MANIFEST
index.html
stylesheet.css
images/logo.png
scripts/main.js
```

Structure of a Manifest File

```
CACHE MANIFEST
# 2010-06-18:v2

# Explicitly cached 'master entries'.
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Resources that require the user to be online.
NETWORK:
login.php
/myapi
http://api.twitter.com

# static.html will be served if main.py is inaccessible
# offline.jpg will be served in place of all images in images/large/
# offline.html will be served in place of all other .html files
FALLBACK:
/main.py /static.html
images/large/ images/offline.jpg
*.html /offline.html
```

Structure of a Manifest File

CACHE:
This is the default section for entries. Files listed under this header (or immediately after the CACHE MANIFEST) will be explicitly cached after they're downloaded for the first time.

NETWORK:
Files listed under this section are white-listed resources that require a connection to the server. All requests to these resources bypass the cache, even if the user is offline. Wildcards may be used.

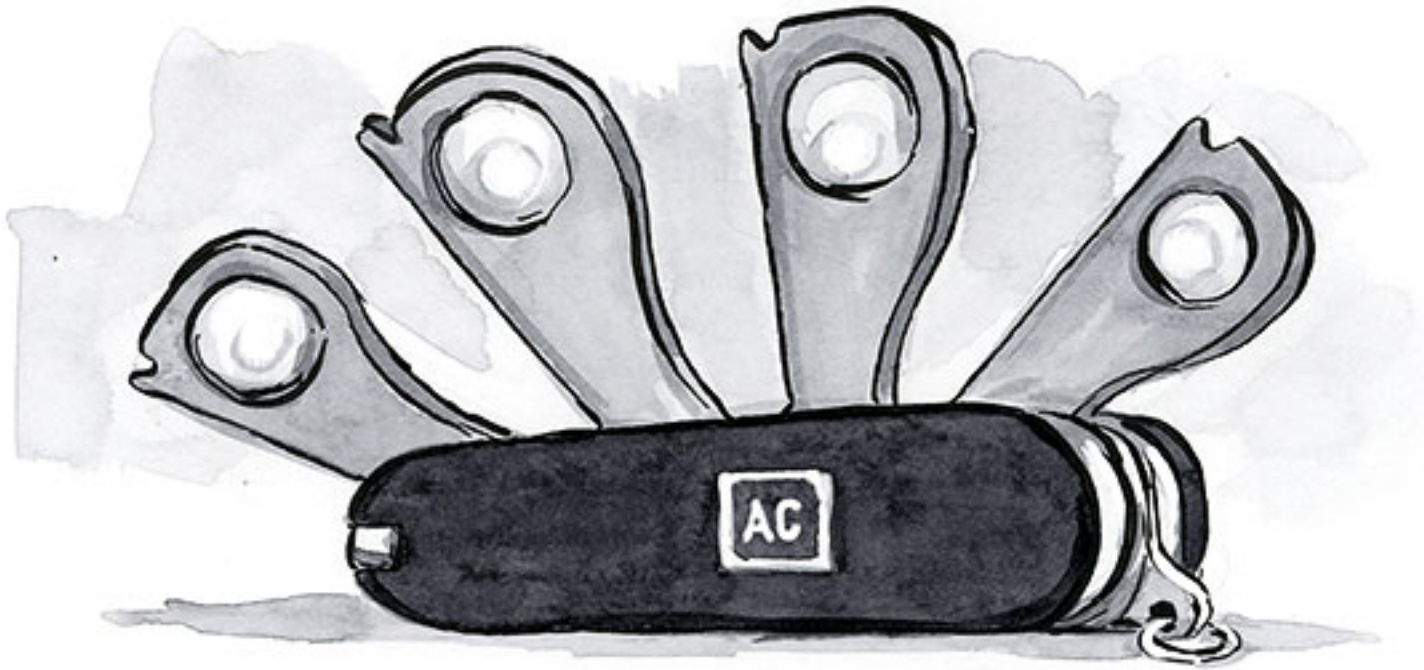
FALLBACK:
An optional section specifying fallback pages if a resource is inaccessible. The first URI is the resource, the second is the fallback. Both URIs must be relative and from the same origin as the manifest file. Wildcards may be used.

Updating the Cache

Once an application is offline it remains cached until one of the following happens:

1. The user clears their browser's data storage for your site.
2. The manifest file is modified. Note: updating a file listed in the manifest doesn't mean the browser will re-cache that resource. The manifest file itself must be altered.
3. The app cache is programatically updated.

Application Cache is a Douchebag



http://www.alistapart.com/articles/application-cache-is-a-douchebag/