Tempo 1.5

Tempo is a tiny JSON rendering engine that enables you to craft data templates in pure HTML.

- Download
- Twitter Example
- Solr Example
- View on GitHub

Why use Tempo?

- Clear separation of concerns: no HTML in your JavaScript files, and no JavaScript in your HTML
- It makes working with AJAX/JSON content a piece of cake
- Works in Safari, Chrome, FireFox, Opera, and Internet Explorer 6+

Key Features

- Itty-bitty footprint, lightning-fast rendering!
- No dependencies - Use with or without jQuery
- Supports nested and conditional templates
- Support for filter and formatting functions and safe attribute setters
- Variable injection for inline JavaScript expressions
- Degrades gracefully if JavaScript is not enabled

1. Include the Tempo script

```
<script src="js/tempo.js" type="text/javascript"></script>

<script>Tempo.prepare("tweets").render(data);</script>
```

2. Compose the data template inline in HTML

```
<ol id="tweets">
    <li data-template>
        <img src="default.png" data-src="{{profile_image_url}}" />
        <h3>{{from_user}}</h3>
        <p>{{text}}<span>, {{created_at|date 'HH:mm on EEEE'}}</span></p>
    </li>
    <li data-template-fallback>Sorry, JavaScript required!</li>
</ol>
```

3. Booyah!

galladryell_

Hijo mío, la felicidad está hecha de pequeñas cosas: Un pequeño yate, una pequeña mansion, una pequeña fortuna... #Groucho Marx, 16:07 on Monday

shendizzles

"The secret of life is honesty and fair dealing. If you can fake that, you've got it made." -Groucho Marx, 16:07 on Monday

Usage

- JSON
- JavaScript
- HTML
- jQuery
- Advanced Topics

JSON

Tempo takes information encoded as JSON and renders it according to an HTML template. Below is a sample array of JSON data.

```
var data = [
    {'name':{'first':'Leonard','last':'Marx'},'nickname':'Chico','born':'March 21, 1887','actor': true,'solo_endeavours':[{'title':'Papa R
    {'name':{'first':'Adolph','last':'Marx'},'nickname':'Harpo','born':'November 23, 1888','actor':true,'solo_endeavours':[{'title':'Too M
    {'name':{'first':'Julius Henry','last':'Marx'},'nickname':'Groucho','born': 'October 2, 1890','actor':true,'solo_endeavours':[{'title'
    {'name':{'first':'Milton','last':'Marx'},'nickname':'Gummo','born':'October 23, 1892'},
    {'name':{'first':'Herbert','last':'Marx'},'nickname':'Zeppo','born':'February 25, 1901','actor':true,'solo_endeavours':[{'title':'A Ki
];
```

JavaScript

Include script

You only need to include one little script.

```
<script src="js/tempo.js" type="text/javascript"></script>
```

Tempo.prepare(element)

To initialize Tempo, run the `prepare()` function to scan an HTML container for data templates, cache them in memory, and remove the data template HTML elements from the page. `Tempo.prepare(element)` returns an instance of a renderer that knows how to layout the data you provide to it.

element

The **ID** of the HTML element (or the element itself) containing your data template. If you're using jQuery, you may pass in a jQuery object instead.

If the container does not contain a default (that is without conditions and not nested) `data-template` the entire contents of the container will be considered to represent the template.

renderer.render()

The Tempo.prepare() function returns an instance of a renderer. Once the JSON data is available, run the `render(data)` function to add the data to the page.

data

The JSON **data** to be rendered. You'll first need to perform an AJAX call to the JSON data source (see below).

```
Tempo.prepare( element ).render( data );
```

```
Tempo.prepare('marx-brothers').render(data);
```

renderer.append()

Renderer methods all return an instance of the renderer (a la fluent) so you can chain calls to it. The `append(data)` function will render the data you pass in and append it to the container.

data

The JSON **data** to append.

```
Tempo.prepare('marx-brothers').render( data ).append( more_brothers );
```

renderer.prepend()

The `prepend(data)` function will render the data you pass in and insert it before others in the container.

data

The JSON **data** to prepend.

```
Tempo.prepare('marx-brothers').render( data ).prepend( brothers_we_didnt_know_about );
```

renderer.clear()

The `clear()` function will empty the container, allowing you to render the data again.

```
Tempo.prepare('marx-brothers').render( data ).clear();
```

HTML

data-template

Any tag with the `data-template` attribute will be flagged as a data template.

For compliance the full (non-minimized) form is also supported: `data-template="data-template"`.

{{fields}}

Any field represented in the JSON data may be retrieved by referencing the field name inside double brackets.

```
<ol id="marx-brothers">
    <li data-template>{{nickname}} {{name.last}}</li>
</ol>
```

The above example would be rendered as:

1. Chico Marx
2. Harpo Marx
3. Groucho Marx
4. Gummo Marx
5. Zeppo Marx

If the JSON data represents an array of arrays (which can not be referenced by field/member name) for example:

var data = [['Leonard','Marx'], ['Adolph','Marx'], ['Julius Henry','Marx'], ['Milton','Marx'], ['Herbert','Marx']];

You can reference array elements with the following notation:

```
<ol id="marx-brothers">
    <li data-template>{{[0]}} {{[1]}}</li>
</ol>
```

The above example would be rendered as:

1. Leonard Marx
2. Adolph Marx
3. Julius Henry Marx
4. Milton Marx
5. Herbert Marx

Nested data-templates

Data templates can even be nested within other data templates. Multiple nested templates are supported.

```
<li data-template>
    {{nickname}} {{name.last}}
    <ul>
        <li data-template="solo_endeavours">{{title}}</li>
    </ul>
</li>
```

1. Chico Marx
   - Papa Romani
2. Harpo Marx
   - Too Many Kisses
   - Stage Door Canteen
3. Groucho Marx
   - Copacabana
   - Mr. Music
   - Double Dynamite
4. Gummo Marx
5. Zeppo Marx
   - A Kiss in the Dark

Conditional Templates

Tempo provides boolean and value-based conditionals, as well as the ability to define multiple data templates per container (the first matching template wins).

```
<ul id="marx-brothers3">
    <li data-template data-if-nickname="Groucho"">{{nickname}} (aka {{name.first}}) was grumpy!</li>
    <li data-template data-if-actor>{{name.first}}, nicknamed '<i>{{nickname}} {{name.last}}</i>' was born on {{born}}</li>

    <!-- Default template -->
    <li data-template>{{name.first}} {{name.last}} was not in any movies!</li>
</ul>
```

- Leonard, nicknamed '*Chico Marx*' was born on March 21, 1887
- Adolph, nicknamed '*Harpo Marx*' was born on November 23, 1888
- Groucho (aka Julius Henry) was grumpy!
- Milton Marx was not in any movies!
- Herbert, nicknamed '*Zeppo Marx*' was born on February 25, 1901

Fallback Template

Use the `data-template-fallback` attribute to identify HTML containers you would like to show if JavaScript is disabled.

To ensure that your data template is not visible before being rendered (either because of JavaScript being disabled or due to latency retrieving the data), it's best practice to hide your templates with CSS. If you add an inline rule of `style="display: none;"` Tempo will simply remove the inline rule once the data has been rendered.

```
<ul id="marx-brothers">
    <li data-template style="display: none;">{{nickname}} {{name.last}}</li>
    <li data-template-fallback>Sorry, JavaScript required!</li>
</ul>
```

If JavaScript is disabled in the browser the above example would be rendered as:

- Sorry, JavaScript required!

Filter and Formatting Functions

Tempo supports a number of filter functions to modify individual values before they are rendered. Filters can be chained to apply multiple ones to the same value.

```
{{ field | default 'default value' }}
```

If the field is undefined, or null, then show the default value specified.

`{{ field | date 'preset or pattern like HH:mm, DD-MM-YYYY' }}`

Creates a date object from the field value, and formats according to presets, or using a date pattern. Available presets are `localedate`, `localetime`, `date` and `time`. The following pattern elements are supported:

- `YYYY or YY`: year as 4 or 2 digits.
- `MMMM, MMM, MM or M`: month of the year as either full name (September), abbreviated (Sep), padded two digit number or simple integer.
- `DD or D`: day of the month.
- `EEEE, EEE or E`: day of the week as either full (Tuesday), abbreviated (Tue) or number.
- `HH or H`: hour of the day.
- `mm or m`: minutes of the hour.
- `ss or s`: seconds.
- `SSS or S`: milliseconds.
- `a`: AM/PM.

If you would like to use any of the format characters verbatim in the pattern then use `\` to escape: `{{ some_date | date '\at HH:mm' }}`. In this case the `a` is not replaced with AM/PM.

`{{ field | replace 'regex_pattern', 'replacement' }}`

Replace all occurrences of the pattern (supports regular expressions) with the replacement. Replacement string can contain backreferences. See Twitter code sample for an example.

`{{ field | trim }}`

Trim any white space at the beginning or end of the value.

`{{ field | upper }}`

Change any lower case characters in the value to upper case.

`{{ field | lower }}`

Change any upper case characters in the value to lower case.

`{{ field | append '&nbsp;some suffix' }}`

If the field value is not empty, then append the string to the value. Helpful if you don't want the suffix to show up in the template if the field is undefined or null. Use &nbsp; if you need before or after the suffix.

`{{ field | prepend 'some prefix&nbsp;' }}`

If the field value is not empty, then prepend the string to the value. Helpful if you don't want the prefix to show up in the template if the field is undefined or null. Use &nbsp; if you need before or after the prefix.

Attribute Setters

If an HTML element attribute in a template is immediately followed by a second attribute with the same name, prefixed with `data-`, then as long as the second one is not empty will the value of the original be replaced with the value of the latter.

In the following example, will the reference to `default_image.png` be replaced by an actual field value if one exists. Notice here that the `.png` suffix is only added if the field is not empty.

```
<img src="default_image.png" data-src="{{actual_image_if_exists | append '.png'}}" ... />
```

Template Tags

Tempo also supports tag blocks.

`{% if javascript-expression %} ... {% endif %}`

The body of the tag will only be rendered if the JavaScript expression evaluates to true.

Variable injection for inline scripts

If you're using scripts inline in your template, you can access JSON object members. You reference these via the __ variable.

```
<a href="#" onclick="alert(__.nickname); return false;">{{name.last}}</a>
```

Similarly you can reference array elements:

```
<a href="#" onclick="alert(__[0]); return false;">{{[0]}}</a>
```

At render time the accessor variable will be replaced by the object it references. If the object is a string then its value will be wrapped in single quotes, otherwise type is preserved.

Using Tempo with jQuery

jQuery's getJSON() method provides an easy means of loading JSON-encoded data from the server using a GET HTTP request.

```
var twitter = Tempo.prepare('tweets');
$.getJSON("http://search.twitter.com/search.json?q='marx brothers'&callback=?", function(data) {
    twitter.render(data.results);
});
```

Advanced Topics

renderer.notify(eventlistener)

After preparing a template you can register an event listener by providing a callback function to be notified of events in the lifecycle. The event listener will be called with a single argument, a `TempoEvent` which has the following properties:

type

The **type** of the event. Constant values are defined in `TempoEvent.Types`.

- `TempoEvent.Types.RENDER_STARTING`: Indicates that rendering has started, or been manually triggered by calling `starting()` on the renderer object.
- `TempoEvent.Types.ITEM_RENDER_STARTING`: Indicates that the rendering of a given individual item is starting.
- `TempoEvent.Types.ITEM_RENDER_COMPLETE`: Indicates that the rendering of a given individual item has completed.
- `TempoEvent.Types.RENDER_COMPLETE`: Indicates that the rendering of all items is completed.

item

The **item** being rendered. This is only available for the `ITEM_RENDER_STARTING` and `ITEM_RENDER_COMPLETE` events.

element

The HTML **element** or template being used for rendering. This is only available for the `ITEM_RENDER_STARTING` and `ITEM_RENDER_COMPLETE` events.

starting()

In some cases you prepare the templates ahead of calling render. In those cases if you would like to e.g. set loader graphics, call the renderer's `starting()` method just prior to issuing e.g. a jQuery request. This will fire the `ITEM_RENDER_STARTING` event.

The following example demonstrates use of both methods above. In this case we prepare the templates, and register our event handler function. The event handler is then notified that the jQuery request is about to be issued (when we manually fire `RENDER_STARTING` with a call to `starting()`) adding a CSS class to the container. We are then notified that rendering is complete so we can remove the CSS class.

```
var twitter = Tempo.prepare('tweets').notify( function (event) {
    if (event.type === TempoEvent.Types.RENDER_STARTING || event.type === TempoEvent.Types.RENDER_COMPLETE) {
        $('#tweets').toggleClass('loading');
    }
});
twitter.starting();
$.getJSON(url, function(data) {
    twitter.render(data.results);
});
```

Binding event handlers

Note that if you've bound event listeners to elements in your template, these will **not** be cloned when the template is used. In order to do this you have two options.

You can either leave binding until the data has been rendered, e.g. by registering a `TempoEvent.Types.RENDER_COMPLETE` listener and doing the binding when that fires, or you can use jQuery live which attaches a handler to elements even though they haven't been created.

Tempo Renderer Information - the `_tempo` variable

You can access information about the rendering process via the `_tempo` variable.

_tempo.index

The `index` variable tells you how many iterations of a given template have been carried out. This is a zero (0) based index. Nested templates have a separate counter.

The following example adds the iteration count to the `class` attribute, prefixed with '`item-`':

```
<ol id="marx-brothers" class="item-{{_tempo.index}}">
    <li data-template>{{nickname}} {{name.last}}</li>
</ol>
```

This example shows how to access the iteration counter using inline JavaScript injection:

```
<a href="#" onclick="alert(__._tempo.index); return false;">{{name.last}}</a>
```

Drag & Drop for WordPress

A revolutionary new way to build pro sites; faster & easier, than ever before.

ads by BSA

via Ad Packs

- Download
- Twitter Example
- Solr Example
- View on GitHub

Brought to you by the friendly guys at TwigKit. Follow Stefan Olafsson on Twitter.