Embedded JavaScript templates

# [![Known Vulnerabilities](https://snyk.io/test/npm/ejs/badge.svg?style=flat)](https://snyk.io/test/npm/ejs)

## Security

Security professionals, before reporting any security issues, please reference the [SECURITY.md](https://github.com/mde/ejs/blob/main/SECURITY.md) in this project, in particular, the following: "EJS is effectively a JavaScript runtime. Its entire job is to execute JavaScript. If you run the EJS render method without checking the inputs yourself, you are responsible for the results."

In short, DO NOT submit 'vulnerabilities' that include this snippet of code:

app.get('/', (req, res) => {

res.render('index', req.query);

});

## Installation

$ npm install ejs

## Features

* Control flow with <% %>
* Escaped output with <%= %> (escape function configurable)
* Unescaped raw output with <%- %>
* Newline-trim mode ('newline slurping') with -%> ending tag
* Whitespace-trim mode (slurp all whitespace) for control flow with <%\_ \_%>
* Custom delimiters (e.g. [? ?] instead of <% %>)
* Includes
* Client-side support
* Static caching of intermediate JavaScript
* Static caching of templates
* Complies with the [Express](http://expressjs.com) view system

## Example

<% if (user) { %>

<h2><%= user.name %></h2>

<% } %>

Try EJS online at: <https://ionicabizau.github.io/ejs-playground/>.

## Basic usage

let template = ejs.compile(str, options);

template(data);

// => Rendered HTML string

ejs.render(str, data, options);

// => Rendered HTML string

ejs.renderFile(filename, data, options, function(err, str){

// str => Rendered HTML string

});

It is also possible to use ejs.render(dataAndOptions); where you pass everything in a single object. In that case, you'll end up with local variables for all the passed options. However, be aware that your code could break if we add an option with the same name as one of your data object's properties. Therefore, we do not recommend using this shortcut.

### Important

You should never give end-users unfettered access to the EJS render method, If you do so you are using EJS in an inherently un-secure way.

### Options

* cache Compiled functions are cached, requires filename
* filename The name of the file being rendered. Not required if you are using renderFile(). Used by cache to key caches, and for includes.
* root Set template root(s) for includes with an absolute path (e.g, /file.ejs). Can be array to try to resolve include from multiple directories.
* views An array of paths to use when resolving includes with relative paths.
* context Function execution context
* compileDebug When false no debug instrumentation is compiled
* client When true, compiles a function that can be rendered in the browser without needing to load the EJS Runtime ([ejs.min.js](https://github.com/mde/ejs/releases/latest)).
* delimiter Character to use for inner delimiter, by default '%'
* openDelimiter Character to use for opening delimiter, by default '<'
* closeDelimiter Character to use for closing delimiter, by default '>'
* debug Outputs generated function body
* strict When set to true, generated function is in strict mode
* \_with Whether or not to use with() {} constructs. If false then the locals will be stored in the locals object. Set to false in strict mode.
* destructuredLocals An array of local variables that are always destructured from the locals object, available even in strict mode.
* localsName Name to use for the object storing local variables when not using with Defaults to locals
* rmWhitespace Remove all safe-to-remove whitespace, including leading and trailing whitespace. It also enables a safer version of -%> line slurping for all scriptlet tags (it does not strip new lines of tags in the middle of a line).
* escape The escaping function used with <%= construct. It is used in rendering and is .toString()ed in the generation of client functions. (By default escapes XML).
* outputFunctionName Set to a string (e.g., 'echo' or 'print') for a function to print output inside scriptlet tags.
* async When true, EJS will use an async function for rendering. (Depends on async/await support in the JS runtime.
* includer Custom function to handle EJS includes, receives (originalPath, parsedPath) parameters, where originalPath is the path in include as-is and parsedPath is the previously resolved path. Should return an object { filename, template }, you may return only one of the properties, where filename is the final parsed path and template is the included content.

This project uses [JSDoc](http://usejsdoc.org/). For the full public API documentation, clone the repository and run jake doc. This will run JSDoc with the proper options and output the documentation to out/. If you want the both the public & private API docs, run jake devdoc instead.

### Tags

* <% 'Scriptlet' tag, for control-flow, no output
* <%\_ 'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
* <%= Outputs the value into the template (escaped)
* <%- Outputs the unescaped value into the template
* <%# Comment tag, no execution, no output
* <%% Outputs a literal '<%'
* %%> Outputs a literal '%>'
* %> Plain ending tag
* -%> Trim-mode ('newline slurp') tag, trims following newline
* \_%> 'Whitespace Slurping' ending tag, removes all whitespace after it

For the full syntax documentation, please see [docs/syntax.md](https://github.com/mde/ejs/blob/master/docs/syntax.md).

### Includes

Includes either have to be an absolute path, or, if not, are assumed as relative to the template with the include call. For example if you are including ./views/user/show.ejs from ./views/users.ejs you would use <%- include('user/show') %>.

You must specify the filename option for the template with the include call unless you are using renderFile().

You'll likely want to use the raw output tag (<%-) with your include to avoid double-escaping the HTML output.

<ul>

<% users.forEach(function(user){ %>

<%- include('user/show', {user: user}) %>

<% }); %>

</ul>

Includes are inserted at runtime, so you can use variables for the path in the include call (for example <%- include(somePath) %>). Variables in your top-level data object are available to all your includes, but local variables need to be passed down.

NOTE: Include preprocessor directives (<% include user/show %>) are not supported in v3.0+.

## Custom delimiters

Custom delimiters can be applied on a per-template basis, or globally:

let ejs = require('ejs'),

users = ['geddy', 'neil', 'alex'];

// Just one template

ejs.render('<p>[?= users.join(" | "); ?]</p>', {users: users}, {delimiter: '?', openDelimiter: '[', closeDelimiter: ']'});

// => '<p>geddy | neil | alex</p>'

// Or globally

ejs.delimiter = '?';

ejs.openDelimiter = '[';

ejs.closeDelimiter = ']';

ejs.render('<p>[?= users.join(" | "); ?]</p>', {users: users});

// => '<p>geddy | neil | alex</p>'

### Caching

EJS ships with a basic in-process cache for caching the intermediate JavaScript functions used to render templates. It's easy to plug in LRU caching using Node's lru-cache library:

let ejs = require('ejs'),

LRU = require('lru-cache');

ejs.cache = LRU(100); // LRU cache with 100-item limit

If you want to clear the EJS cache, call ejs.clearCache. If you're using the LRU cache and need a different limit, simple reset ejs.cache to a new instance of the LRU.

### Custom file loader

The default file loader is fs.readFileSync, if you want to customize it, you can set ejs.fileLoader.

let ejs = require('ejs');

let myFileLoad = function (filePath) {

return 'myFileLoad: ' + fs.readFileSync(filePath);

};

ejs.fileLoader = myFileLoad;

With this feature, you can preprocess the template before reading it.

### Layouts

EJS does not specifically support blocks, but layouts can be implemented by including headers and footers, like so:

<%- include('header') -%>

<h1>

Title

</h1>

<p>

My page

</p>

<%- include('footer') -%>

## Client-side support

Go to the [Latest Release](https://github.com/mde/ejs/releases/latest), download ./ejs.js or ./ejs.min.js. Alternately, you can compile it yourself by cloning the repository and running jake build (or $(npm bin)/jake build if jake is not installed globally).

Include one of these files on your page, and ejs should be available globally.

### Example

<div id="output"></div>

<script src="ejs.min.js"></script>

<script>

let people = ['geddy', 'neil', 'alex'],

html = ejs.render('<%= people.join(", "); %>', {people: people});

// With jQuery:

$('#output').html(html);

// Vanilla JS:

document.getElementById('output').innerHTML = html;

</script>

### Caveats

Most of EJS will work as expected; however, there are a few things to note:

1. Obviously, since you do not have access to the filesystem, ejs.renderFile() won't work.
2. For the same reason, includes do not work unless you use an include callback. Here is an example:

let str = "Hello <%= include('file', {person: 'John'}); %>",

fn = ejs.compile(str, {client: true});

fn(data, null, function(path, d){ // include callback

// path -> 'file'

// d -> {person: 'John'}

// Put your code here

// Return the contents of file as a string

}); // returns rendered string

See the [examples folder](https://github.com/mde/ejs/tree/master/examples) for more details.

## CLI

EJS ships with a full-featured CLI. Options are similar to those used in JavaScript code:

* -o / --output-file FILE Write the rendered output to FILE rather than stdout.
* -f / --data-file FILE Must be JSON-formatted. Use parsed input from FILE as data for rendering.
* -i / --data-input STRING Must be JSON-formatted and URI-encoded. Use parsed input from STRING as data for rendering.
* -m / --delimiter CHARACTER Use CHARACTER with angle brackets for open/close (defaults to %).
* -p / --open-delimiter CHARACTER Use CHARACTER instead of left angle bracket to open.
* -c / --close-delimiter CHARACTER Use CHARACTER instead of right angle bracket to close.
* -s / --strict When set to true, generated function is in strict mode
* -n / --no-with Use 'locals' object for vars rather than using with (implies --strict).
* -l / --locals-name Name to use for the object storing local variables when not using with.
* -w / --rm-whitespace Remove all safe-to-remove whitespace, including leading and trailing whitespace.
* -d / --debug Outputs generated function body
* -h / --help Display this help message.
* -V/v / --version Display the EJS version.

Here are some examples of usage:

$ ejs -p [ -c ] ./template\_file.ejs -o ./output.html

$ ejs ./test/fixtures/user.ejs name=Lerxst

$ ejs -n -l \_ ./some\_template.ejs -f ./data\_file.json

### Data input

There is a variety of ways to pass the CLI data for rendering.

Stdin:

$ ./test/fixtures/user\_data.json | ejs ./test/fixtures/user.ejs

$ ejs ./test/fixtures/user.ejs < test/fixtures/user\_data.json

A data file:

$ ejs ./test/fixtures/user.ejs -f ./user\_data.json

A command-line option (must be URI-encoded):

./bin/cli.js -i %7B%22name%22%3A%20%22foo%22%7D ./test/fixtures/user.ejs

Or, passing values directly at the end of the invocation:

./bin/cli.js -m $ ./test/fixtures/user.ejs name=foo

### Output

The CLI by default send output to stdout, but you can use the -o or --output-file flag to specify a target file to send the output to.

## IDE Integration with Syntax Highlighting

VSCode:Javascript EJS by *DigitalBrainstem*

## Related projects

There are a number of implementations of EJS:

* TJ's implementation, the v1 of this library: <https://github.com/tj/ejs>
* EJS Embedded JavaScript Framework on Google Code: <https://code.google.com/p/embeddedjavascript/>
* Sam Stephenson's Ruby implementation: <https://rubygems.org/gems/ejs>
* Erubis, an ERB implementation which also runs JavaScript: <http://www.kuwata-lab.com/erubis/users-guide.04.html#lang-javascript>
* DigitalBrainstem EJS Language support: <https://github.com/Digitalbrainstem/ejs-grammar>

## License

Licensed under the Apache License, Version 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>)

EJS Embedded JavaScript templates copyright 2112 [mde@fleegix.org](mailto:mde@fleegix.org).