EJS

Get Started

Install

It's easy to install EJS with NPM.

```
$ npm install ejs
```

Use

Pass EJS a template string and some data. BOOM, you've got some HTML.

```
let ejs = require('ejs');
let people = ['geddy', 'neil', 'alex'];
let html = ejs.render('<%= people.join(", "); %>', {people: peop
```

CLI

Feed it a template file and a data file, and specify an output file.

```
ejs ./template_file.ejs -f data_file.json -o ./output.html
```

Browser support

Download a browser build from the latest release (https://github.com/mde/ejs/releases/latest), and use it in a script tag.

```
<script src="ejs.js"></script>
<script>
  let people = ['geddy', 'neil', 'alex'];
  let html = ejs.render('<%= people.join(", "); %>', {people: pe
</script>
```

EJS

Docs

Example

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

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
});
```

Options

- `cache` Compiled functions are cached, requires `filename`
- `filename` Used by `cache` to key caches, and for includes
- `root` Set project root 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` Returns standalone compiled function
- `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. (Implies `--strict`)

- `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.

Tags

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

Includes

Includes are relative to the template with the `include` call. (This requires the 'filename' option.) For example if you have "./views/users.ejs" and "./views/user/show.ejs" you would use `<%- include('user/show'); %>`.

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>
```

CLI

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

- `cache` Compiled functions are cached, requires `filename`
- `-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.

Some examples of use:

```
$ 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
```

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('<?= users.join(" | "); ?>', {users: users},
    {delimiter: '?'});
// => 'geddy | neil | alex'

// Or globally
ejs.delimiter = '$';
ejs.render('<$= users.join(" | "); $>', {users: users});
// => 'geddy | neil | alex'
```

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 myFileLoader = function (filePath) {
  return 'myFileLoader: ' + fs.readFileSync(filePath);
};

ejs.fileLoader = myFileLoader;
```

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:

EJS

```
<%- 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: pe
  // 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:

EJS

```
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
```

Using EJS with Express

This GitHub Wiki page (https://github.com/mde/ejs/wiki/Using-EJS-with-Express) explains various ways of passing EJS options to Express.

Support

Stack Overflow (http://stackoverflow.com/questions/tagged/ejs)

Ask questions (http://stackoverflow.com/questions/ask) about specific problems you have faced, including details about what exactly you are trying to do. Make sure you tag your question with `ejs`. You can also read through existing ejs questions (http://stackoverflow.com/questions/tagged/ejs).

GitHub issues (https://github.com/mde/ejs/issues)

The issue tracker (https://github.com/mde/ejs/issues) is the preferred channel for bug reports, features requests and submitting pull requests.

License