API Reference

This page details how to render Pug using the JavaScript API.

Tip

Pug is available in your Web browser's console! To test drive Pug's API, as documented on this page, try entering:

```
pug.render('p Hello world!');
```

Options ¶

All API methods take the following set of options:

`filename: string`
The name of the file being compiled. Used in exceptions, and required for relative includes and extends. Defaults to `'Pug'`.

`basedir: string`
The root directory of all absolute inclusion.

`doctype: string`
If the doctype is not specified as part of the template, you can specify it here. It is sometimes useful to get self-closing tags and remove mirroring of boolean attributes; see doctype documentation for more information.

`pretty: boolean | string`
Adds whitespace to the resulting HTML to make it easier for a human to read using `'  '` as indentation. If a string is specified, that will be used as indentation instead (e.g. `'\t'`). Defaults to `false`.

`filters: object`
Hash table of custom filters. Defaults to `undefined`.

`self: boolean`
Use a `self` namespace to hold the locals. It will speed up the compilation, but instead of writing `variable` you will have to write `self.variable` to access a property of the locals object. Defaults to `false`.

`debug: boolean`
If set to `true`, the tokens and function body are logged to stdout.

`compileDebug: boolean`

If set to `true`, the function source will be included in the compiled template for better error messages (sometimes useful in development). It is enabled by default unless used with Express in production mode.

`globals: Array<string>`

Add a list of global names to make accessible in templates.

`cache: boolean`

If set to `true`, compiled functions are cached. `filename` must be set as the cache key. Only applies to `render` functions. Defaults to `false`.

`inlineRuntimeFunctions: boolean`

Inline runtime functions instead of `require`-ing them from a shared version. For `compileClient` functions, the default is `true` so that one does not have to include the runtime. For all other compilation or rendering types, the default is `false`.

`name: string`

The name of the template function. Only applies to `compileClient` functions. Defaults to `'template'`.

Methods ¶

pug.compile(source, ?options) ¶

Compile a Pug template to a function which can be rendered multiple times with different locals.

`source: string`

The source Pug template to compile

`options: ?options`

An options object

`returns: function`

A function to generate the HTML from an object containing locals

```
var pug = require('pug');

// Compile a function
var fn = pug.compile('string of pug', options);

// Render the function
var html = fn(locals);
// => '<string>of pug</string>'
```

pug.compileFile(path, ?options) ¶

Compile a Pug template from a file to a function which can be rendered multiple times with different locals.

path: string

The path to a Pug file

options: ?options

An options object

returns: function

A function to generate the HTML from an object containing locals

```
var pug = require('pug');

// Compile a function
var fn = pug.compileFile('path to pug file', options);

// Render the function
var html = fn(locals);
// => '<string>of pug</string>'
```

pug.compileClient(source, ?options) ¶

Compile a Pug template to a string of JavaScript that can be used client side along with the Pug runtime.

source: string

The Pug template to compile

options: ?options

An options object

returns: string

A string of JavaScript representing a function

```
var pug = require('pug');

// Compile a function
var fn = pug.compileClient('string of pug', options);

// Render the function
var html = fn(locals);
// => 'function template(locals) { return "<string>of
pug</string>"; }'
```

pug.compileClientWithDependenciesTracked ?options) ¶

Same as `compileClient` except that this method returns an object of the form:

```
{
  'body': 'function (locals) {...}',
  'dependencies': ['filename.pug']
}
```

You should only use this method if you need `dependencies` to implement something like watching for changes to the Pug files.

pug.compileFileClient(path, ?options) ¶

Compile a Pug template file to a string of JavaScript that can be used client side along with the Pug runtime.

`path: string`
The path to a Pug file

`options: ?options`
An options object

`options.name: string`
If you pass a `.name` property on the options object, it will be used as the function name for your client side template function.

`returns: string`
A JavaScript function body.

First, our template file.

```
h1 This is a Pug template
h2 By #{author}
```

Then, we compile the Pug file into a function string.

```
var fs = require('fs');
var pug = require('pug');

// Compile the template to a function string
var jsFunctionString =
pug.compileFileClient('/path/to/pugFile.pug', {name:
"fancyTemplateFun"});

// Maybe you want to compile all of your templates to a
templates.js file and serve it to the client
fs.writeFileSync("templates.js", jsFunctionString);
```

Here's what the output function string looks like (written to `templates.js`).

```
function fancyTemplateFun(locals) {
  var buf = [];
  var pug_mixins = {};
  var pug_interp;

  var locals_for_with = (locals || {});

  (function (author) {
    buf.push("<h1>This is a Pug template</h1><h2>By "
      + (pug.escape((pug_interp = author) == null ? '' :
pug_interp))
      + "</h2>");
  }.call(this, "author" in locals_for_with ?
    locals_for_with.author : typeof author !== "undefined" ?
      author : undefined)
  );

  return buf.join("");
}
```

Be sure to send the Pug runtime (`node_modules/pug/runtime.js`) to the client in addition to the template that you just compiled.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="/runtime.js"></script>
    <script src="/templates.js"></script>
  </head>

  <body>
    <h1>This is one fancy template.</h1>

    <script type="text/javascript">
      var html = window.fancyTemplateFun({author: "enlore"});
      var div = document.createElement("div");
      div.innerHTML = html;
      document.body.appendChild(div);
    </script>
  </body>
</html>
```

pug.render(source, ?options, ?callback) ¶

source: string

The source Pug template to render

options: ?options

An options object, also used as the locals object

`callback: ?function`

Node.js-style callback receiving the rendered results. **This callback is called synchronously.**

`returns: string`

The resulting HTML string

```
var pug = require('pug');

var html = pug.render('string of pug', options);
// => '<string>of pug</string>'
```

pug.renderFile(path, ?options, ?callback) ¶

`path: string`

The path to the Pug file to render

`options: ?options`

An options object, also used as the locals object

`callback: ?function`

Node.js-style callback receiving the rendered results. **This callback is called synchronously.**

`returns: string`

The resulting HTML string

```
var pug = require('pug');

var html = pug.renderFile('path/to/file.pug', options);
// ...
```

Properties ¶

pug.filters ¶

A hash table of custom filters.

This object has the same semantics as the `filters` option, but applies globally to all Pug compilation. When a filter is present in both `pug.filters` and `options.filters`, the `filters` option takes precedence.

Deprecated

This property has been deprecated in favor of the `filters` option.