Setting up Emacs for JavaScript (part #1)

Nicolas Petton Apr 23, 2017

There's a lot that can be done to make Emacs a great tool for JavaScript development. So much that I had to split it up into several posts.

In this first article we'll see how to setup js2-mode and two other packages that rely upon it: js2-refactor and xref-js2.

Emacs comes with a major mode for JavaScript named `js-mode`. While it is a good major mode, we'll be using `js2-mode` instead, an external package that extends `js-mode` and provides a very interesting feature: instead of using regular expressions, it parses buffers and builds an AST for things like syntax highlighting. While diverging a bit from the traditional "Emacs way of doing things", this is really interesting, and used as the foundation for other features like refactorings.

Setting up js2-mode

If you haven't done it already, you should first setup `package.el` to use MELPA, then install and setup `js2-mode` like the following:

```
M-x package-install RET js2-mode RET
```

```
(require 'js2-mode)
(add-to-list 'auto-mode-alist '("\\.js\\'" . js2-mode))

;; Better imenu
(add-hook 'js2-mode-hook #'js2-imenu-extras-mode)
```

js2-refactor and xref-js2

Now that we're using `js2-mode` for JavaScript buffers, let's take advantage its capabilities of and install two other packages: js2-refactor and xref-js2.

`js2-refactor` adds powerful refactorings based on the AST generated by `js2-mode`, and `xref-js2` makes it easy to jump to function references or definitions.

`xref-js2` uses `ag` to perform searches, so you'll need to install it as well.

```
M-x package-install RET js2-refactor RET
M-x package-install RET xref-js2 RET
```

```
(require 'js2-refactor)
(require 'xref-js2)

(add-hook 'js2-mode-hook #'js2-refactor-mode)
(js2r-add-keybindings-with-prefix "C-c C-r")
(define-key js2-mode-map (kbd "C-k") #'js2r-kill)

;; js-mode (which js2 is based on) binds "M-." which conflicts with xref, so
;; unbind it.
(define-key js-mode-map (kbd "M-.") nil)

(add-hook 'js2-mode-hook (lambda ()
  (add-hook 'xref-backend-functions #'xref-js2-xref-backend nil t)))
```

Now that everything's setup, let's see how to use `js2-refactor` and `xref-js2`.

Using js2-refactor

`js2-refactor` is a JavaScript refactoring library for emacs.

It provides a collection of refactoring functions leveraging the AST provided by `js2-mode`.

Refactorings go from inlining/extracting variables to converting ternary

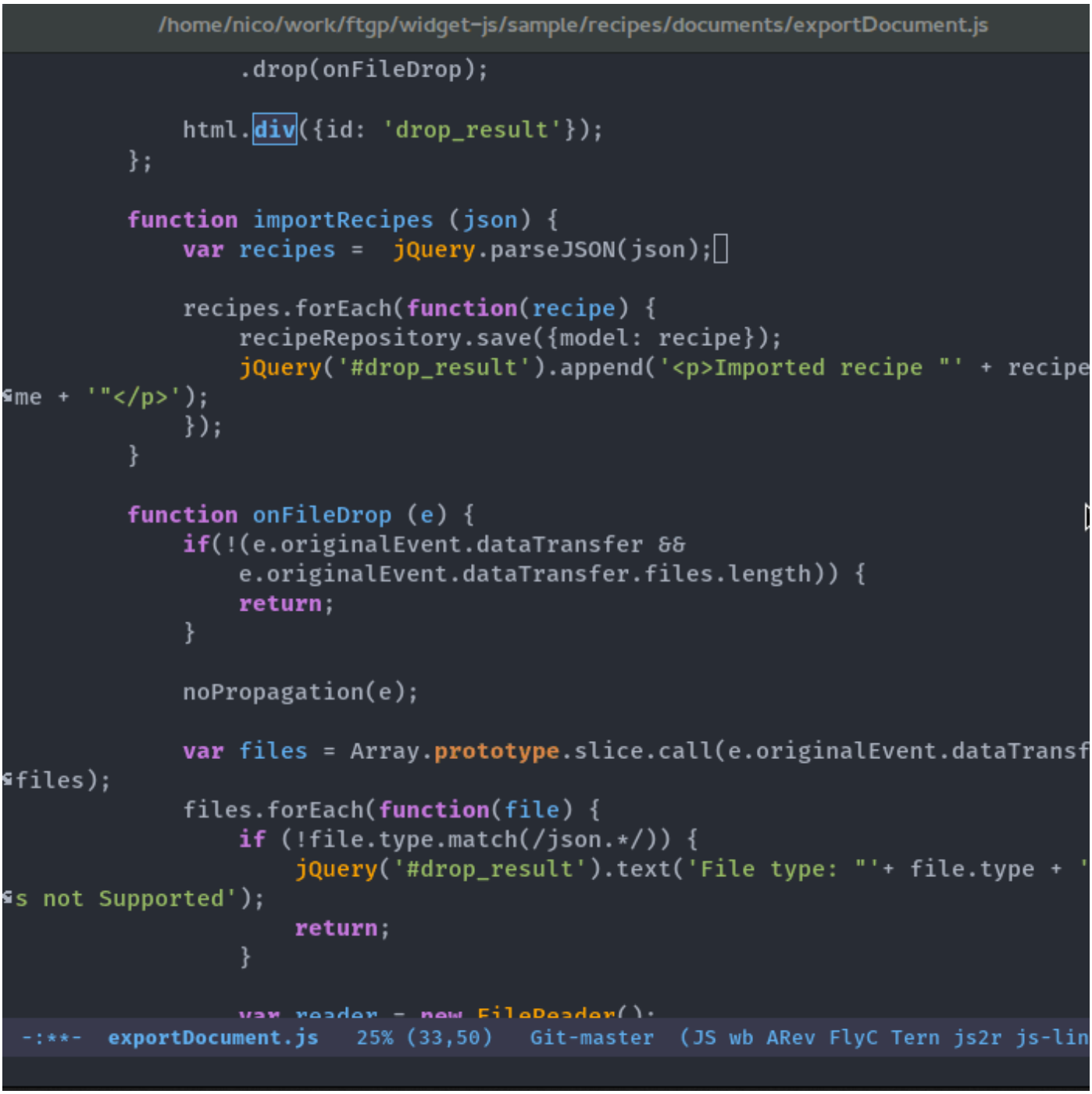
operators to if statements. The README provides the full list of keybindings.

One minor tweak that I really couldn't live without is binding `js2r-kill` to `C-k` in JS buffers:

```
(define-key js2-mode-map (kbd "C-k") #'js2r-kill)
```

This command is very similar to killing in `paredit`: It kills up to the end of the line, but always keeping the AST valid.

Here's a usage example of `js2-refactor`: renaming a function parameter and inlining a variable.



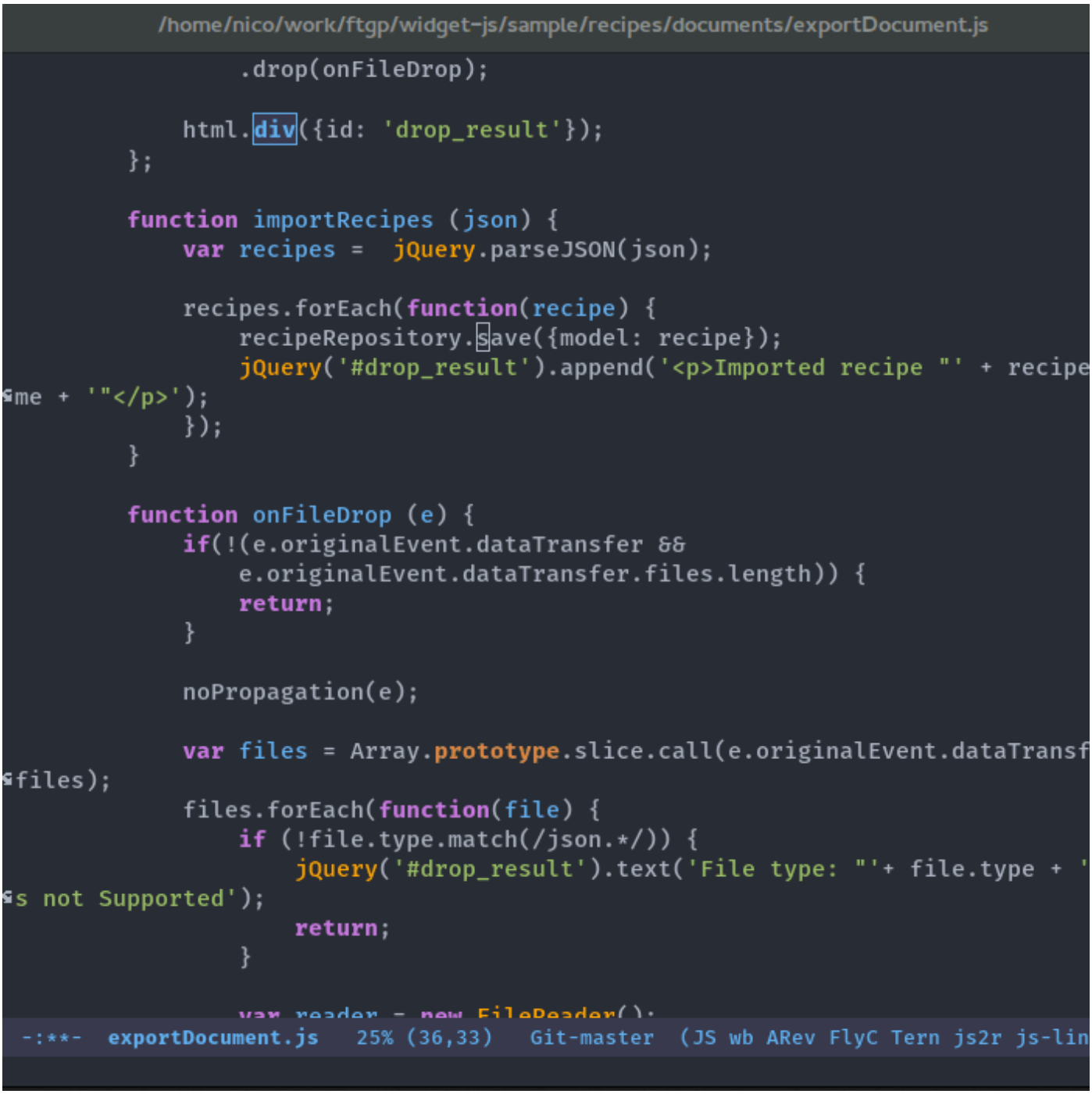
Using xref-js2

`xref-js2` adds support for quickly jumping to function definitions or references to JavaScript projects in Emacs (>= 25.1).

Instead of using a tag system, it relies on `ag` to query the codebase of a project.

- `M-.` Jump to definition
- `M-?` Jump to references
- `M-,` Pop back to where `M-.` was last invoked.

Here's a usage example of `xref-js2`:



Until next time

You should now have a decent setup for `js2-mode` and associated tools.

We still have a lot to explore like linting, getting good auto-completion, using snippets, setting up a REPL and debugger, etc. but I promised I would keep posts short, so stay tuned for part #2!