An Intelligent Python IDE With Emacs, Projectile, and Jedi

Drew Werner

May 5, 2014

Roadmap

- Motivating demo
- Installation guide
- Customization tips

Try it out: github.com/wernerandrew/jedi-starter

An Emacs IDE

- Easily find files and switch between projects
- Easy navigation through function definitions
- Contextual documentation
- Inline help for complex function calls

How it all works

- Need an interface for completion and showing alternatives.
  - `auto-complete`
- Need an interpreter that knows the language
  - Here, a server that wraps a Python parsing library
- You need glue.
  - `epc`, `jedi.el`

All this involves some .emacs customization.

The Jedi Starter VM

- You'll need:
  - Some unix-y CLI environment (OS X, Linux)
  - The `jedi-starter` repo (on Github)
  - Vagrant (tested with 1.4.3)
  - VirtualBox (tested with 4.2.16)
- After cloning, from a terminal:
  - `cd jedi-starter`
  - `vagrant up`
  - `vagrant ssh`
- Initialization code is in `jedi-starter.el`

Structure of jedi-starter.el

```
(add-to-list 'load-path "~/.emacs.d")
(require 'package)

;; Package setup code goes here

;; Global config vars, global helper functions

(add-hook
 'after-init-hook
 '(lambda ()
    ;; Package specific initialization goes here
 ))
```

Installation

- You'll need:
  - `projectile`
  - `auto-complete`
  - `epc`
  - `jedi`
- Manual installation is possible, but annoying
- `package.el` is much better

Package Management

- Built into Emacs 24, available for Emacs 23.
  - (See Github repo for Emacs 23 example.)

```
(require 'package)
(package-initialize)
(add-to-list
 'package-archives
 '("melpa" . "http://melpa.milkbox.net/packages/"))
```

- One gotcha: don't forget the trailing "/" on the URL.
- Packages are stored in `/.emacs.d/elpa/`

Getting the packages

- One way: `M-x list-packages`
  - Gives you a nice interface
  - But hard to reproduce your environment
- A better way: Use the `package.el` API

Simple auto-installation

```
(defvar local-packages '(projectile auto-complete epc jedi))

(defun uninstalled-packages (packages)
  (delq nil
        (mapcar (lambda (p)
                  (if (package-installed-p p nil) nil p))
                packages)))

(let ((need-to-install
       (uninstalled-packages local-packages)))
  (when need-to-install
    (progn
      (package-refresh-contents)
      (dolist (p need-to-install)
        (package-install p)))))
```

Working with Projectile

- Some helpful commands:

C-c p s Switch to project
C-c p f List files in project
C-c p k Kill all buffers for project

- More info: https://github.com/bbatsov/projectile

Easy setup:

```
(require 'projectile)
(projectile-global-mode)

;; Emacs 23 hack
(when (not (boundp 'remote-file-name-inhibit-cache))
  (setq remote-file-name-inhibit-cache t))
```

Package Config

`auto-complete` is also easy:

```
(require 'auto-complete-config)
(ac-config-default)

;; If you really like the menu
(setq ac-show-menu-immediately-on-auto-complete t)
```

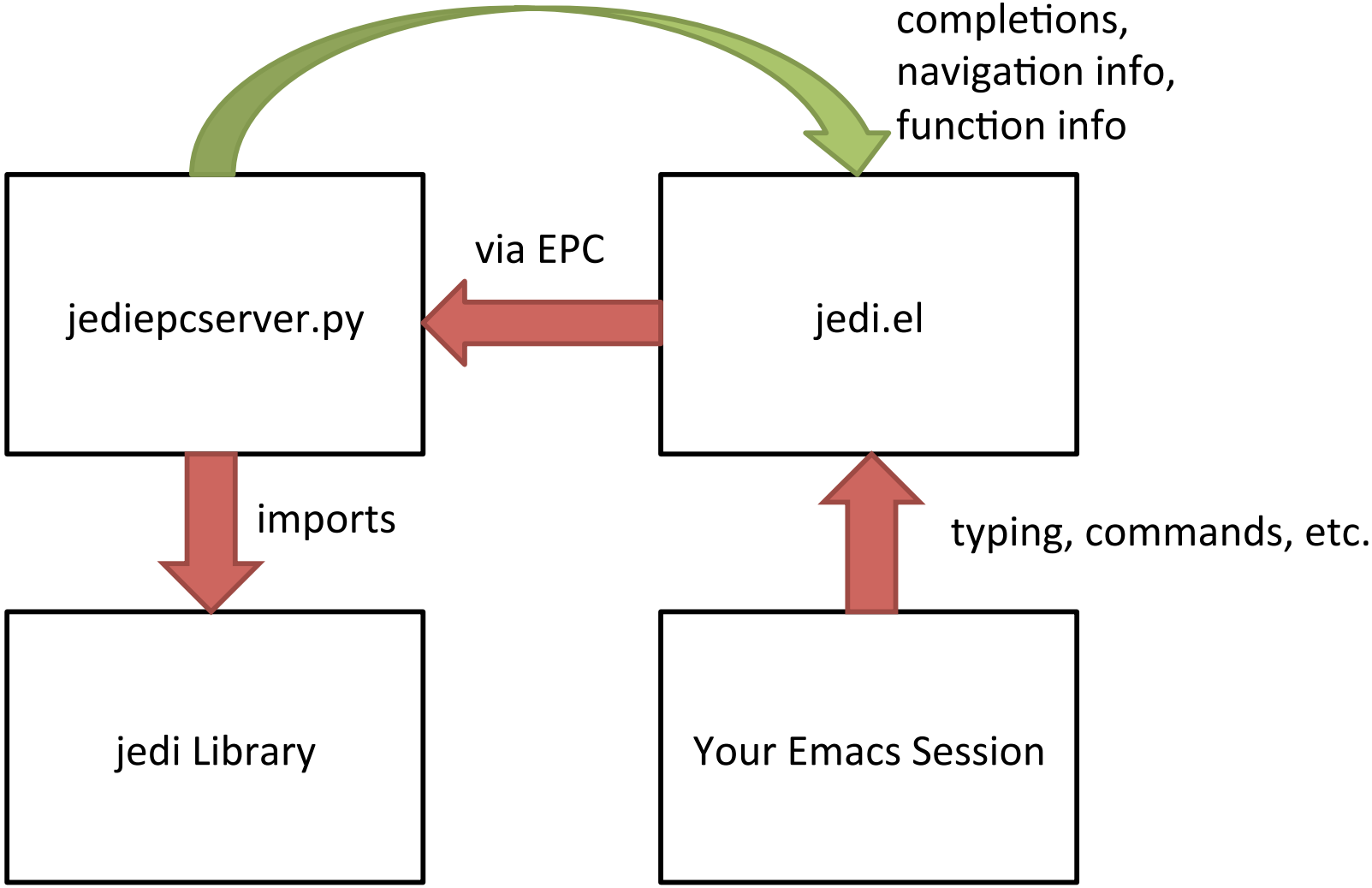
- Automatically integrates with most common programming modes
- But only enables basic completions
  - Language-specific keywords
  - Words in buffers with same mode
- Doesn't know anything about syntax

Jedi: The Brains

Several things have to play nicely for this all to work:

- Jedi
  - A Python library for contextual parsing of files
  - Not specific to Emacs
  - https://github.com/davidhalter/jedi
- EPC
  - Simple Emacs/Python RPC library
- Jedi.el
  - Small Python server wrapping some Jedi features
  - Elisp front end to the server

Jedi Components



Jedi Dependencies

Jedi depends on some Python packages not installed by package.el.
You have two choices:

- Let Jedi handle it
  - Requires virtualenv and pip to be installed
  - A one-time `M-x jedi:install-server`
  - Dependencies are installed in sandboxed environment
  - Doesn't work (yet) with other package managers (e.g. conda)
- Do it yourself
  - Necessary if you can't use virtualenv/pip
  - Install `epc` and `jedi` python modules globally
  - Need to ensure they are available to Jedi server
  - May need to point Jedi to a particular installed Python

Configuration

The bare minimum:

```
(require 'jedi)
;; Hook up to autocomplete
(add-to-list 'ac-sources 'ac-source-jedi-direct)
;; Enable for python-mode
(add-hook 'python-mode-hook 'jedi:setup)
```

Jedi Server Options

- Finding your project (`--sys-path`)
- Finding your installed modules (`--virtual-env`)
  - Note that the active virtualenv can be found automatically
- Details: `C-h v <RET> jedi:server-args <RET>`

Configuration (Simplified)

```
(defvar jedi-config:with-virtualenv nil
  "Set to non-nil to point to a particular virtualenv.")

;; Variables to help find the project root
(defvar jedi-config:vcs-root-sentinel ".git")
(defvar jedi-config:python-module-sentinel "__init__.py")

;; Function to find project root given a buffer
(defun get-project-root (buf repo-type init-file)
  (vc-find-root (expand-file-name (buffer-file-name buf)) repo-type))

(defvar jedi-config:find-root-function 'get-project-root)

;; And call this on initialization
(defun current-buffer-project-root ()
  (funcall jedi-config:find-root-function
           (current-buffer)
           jedi-config:vcs-root-sentinel
           jedi-config:python-module-sentinel))
```

(See online for fancier, more robust version.)

Set the server args

- A list: `(ARG1 VALUE1 ARG2 VALUE2 ...)`
- Store in buffer local variable `jedi:server-args`

```
(defun jedi-config:setup-server-args ()
  ;; little helper macro
  (defmacro add-args (arg-list arg-name arg-value)
    `(setq ,arg-list (append ,arg-list (list ,arg-name ,arg-value))))

  (let ((project-root (current-buffer-project-root)))

    ;; Variable for this buffer only
    (make-local-variable 'jedi:server-args)

    ;; And set our variables
    (when project-root
      (add-args jedi:server-args "--sys-path" project-root))
    (when jedi-config:with-virtualenv
      (add-args jedi:server-args "--virtual-env"
                                 jedi-config:with-virtualenv))))
```

Sidebar: Finding Python

If you can't use virtualenv, you might need to explicitly select a Python to run.

Also, on Mac OS X (and perhaps other GUI environments), your PATH may need to be set explicitly.

```
(defvar jedi-config:use-system-python t)

(defun jedi-config:set-python-executable ()
 (set-exec-path-from-shell-PATH) ;; for OS X
 (make-local-variable 'jedi:server-command)
 (set 'jedi:server-command
      (list (executable-find "python")
            (cadr default-jedi-server-command))))
```

Putting everything together

```
(add-hook 'python-mode-hook
          'jedi-config:setup-server-args)

(when jedi-config:use-system-python
  (add-hook 'python-mode-hook
            'jedi-config:set-python-executable))
```

Some useful commands

Description	Default	Suggest
`jedi:goto-definition` Move to definition of symbol at point	`C-c .`	`M-.`
`jedi:goto-definition-pop-marker` Move to previous location of point	`C-c ,`	`M-,`
`jedi:show-doc` Show docstring for symbol at point in new window	`C-c ?`	`M-?`
`jedi:get-in-function-call` Pop up signature for function at point	None	`M-/`

Local Jedi keybindings

```
(defun jedi-config:setup-keys ()
  (local-set-key (kbd "M-.") 'jedi:goto-definition)
  (local-set-key (kbd "M-,") 'jedi:goto-definition-pop-marker)
  (local-set-key (kbd "M-?") 'jedi:show-doc)
  (local-set-key (kbd "M-/") 'jedi:get-in-function-call))

(add-hook 'python-mode-hook 'jedi-config:setup-keys)
```

Jedi Miscellany

- Small hack to never show in-function call automatically:
  - `(setq jedi:get-in-function-call-delay 10000000)`
  - Recommended if you bind this to a key
- Complete when you type a dot:
  - `(setq jedi:complete-on-dot t)`
  - Useful when typing method calls

Other Packages

Various other packages use similar techniques to provide intelligent documentation and completion.

- `robe` (Ruby)
- `irony-mode` (C / C++)
- `gocode` (golang)
- CEDET (Various)
- Others?

Questions?

Happy editing!
github.com/wernerdrew/jedi-starter
@wernerdrew