



Clojure

RECIPES

Julian GAMBLE

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Clojure Recipes

This page intentionally left blank

Clojure Recipes

Julian Gamble

◆◆Addison-Wesley

New York • Boston • Indianapolis • San Francisco
Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Gamble, Julian, author.

Clojure recipes / Julian Gamble.

pages cm

Includes index.

ISBN 978-0-321-92773-6 (pbk. : alk. paper)—ISBN 0-321-92773-7

1. Clojure (Computer program language) 2. Software patterns. I. Title.

QA76.73.C565G36 2015

005.1—dc23

2015028332

Copyright © 2016 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, 200 Old Tappan Road, Old Tappan, New Jersey 07675, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-92773-6

ISBN-10: 0-321-92773-7

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana. First printing, October 2015



*To my amazing wife Jo-Ann . . .
. . . You Rock!*



This page intentionally left blank

Contents

Preface	xv
1 Starting Your Project with Leiningen	1
Assumptions	1
Benefits	1
The Recipe—Windows	1
The Recipe—Mac	3
Conclusion	4
Postscript—Setting Up a JDK on a Mac	4
Postscript—Setting Up a JDK on Windows	6
2 Packaging Clojure for a Java EE Environment	11
Assumptions	11
Benefits	11
The Recipe—Common	11
Conclusion	14
Postscript—Setting Up Tomcat on a Mac	14
Postscript—Setting Up Tomcat on Windows	16
3 Creating a REST Server in Compojure	19
Assumptions	19
Benefits	19
The Recipe—Code	19
Testing the Solution	20
Notes on the Recipe	22
Conclusion	23
4 Creating a REST Server with Liberator	25
Assumptions	25
Benefits	25
The Recipe—Code	25
Testing the Solution	28
Notes on the Recipe	29

Context	32
Origins	33
REST Hypermedia	33
Conclusion	33
5 A REST Client in ClojureScript	35
Assumptions	35
Benefits	35
The Recipe—Code	36
Testing the Solution	38
Notes on the Recipe	39
Conclusion	40
6 A Simple JSON Server	41
Assumptions	41
Benefits	41
The Recipe—Code	41
Testing the Solution	44
Notes on the Recipe	45
Conclusion	47
7 A Simple Server Using the Pedestal Framework	49
Assumptions	49
Benefits	49
Context	49
The Recipe—Code	50
Testing the Solution	52
Notes on the Recipe	53
Conclusion	54
8 A Stock Ticker on the Pedestal Framework Server	55
Assumptions	55
Benefits	55
The Recipe—Code	55
Testing the Solution	60
Notes on the Recipe	62
Conclusion	67

9	Simplifying Logging with a Macro	69
	Assumptions	69
	Benefits	69
	The Recipe—Code	70
	Testing the Solution	72
	Notes on the Recipe	74
	Conclusion	76
10	Extending the Compiler with a Macro	79
	Assumptions	79
	Benefits	79
	The Recipe—Code	79
	Testing the Solution	81
	Notes on the Recipe	82
	Conclusion	85
11	Simplifying Datomic Syntax by Writing a DSL	87
	Assumptions	87
	Benefits	87
	The Recipe—Code	88
	Testing the Solution	97
	Testing Create Schema	97
	Testing Add Datom	98
	Testing Create Nested Schema	100
	Testing Add Nested Datom	101
	Notes on the Recipe	103
	create.clj	103
	create_test.clj	104
	add.clj	104
	add_test.clj	104
	create_nested.clj	104
	create_nested_test.clj	105
	add_nested.clj	105
	add_nested_test.clj	106
	Conclusion	106

12	Reading the SASS DSL and Generating CSS with Clojure Zippers	107
	Assumptions	107
	Benefits	107
	Outline—Features of SASS	107
	The Recipe—Code	109
	Testing the Solution	121
	Notes on the Recipe	122
	Conclusion	133
13	Introduction to Cascalog	135
	Assumptions	135
	Benefits	135
	The Recipe—Code	135
	Testing the Solution	137
	Notes on the Recipe	139
	Conclusion	141
14	Cascalog and Hadoop	143
	Assumptions	143
	Benefits	143
	The Recipe—Code	143
	Testing the Solution	145
	Conclusion	146
	Postscript—Setting Up Hadoop on a Mac	146
	Postscript—Setting Up Hadoop on a Windows Machine	147
15	Loading a Data File into Cascalog	149
	Assumptions	149
	Benefits	149
	The Recipe—Code	149
	Testing the Solution	150
	Conclusion	151
16	Writing Out a Data File with Cascalog	153
	Assumptions	153
	Benefits	153

The Recipe—Code	153
Testing the Solution	154
Notes on the Recipe	155
Conclusion	155
17 Cascalog and Structured Data	157
Assumptions	157
Benefits	157
The Recipe—Code	157
Testing the Recipe	159
Notes on the Solution	160
Conclusion	161
18 Loading Custom Data Formats into Cascalog	163
Assumptions	163
Benefits	163
The Recipe—Code	163
Testing the Recipe	175
Notes on the Solution	176
Conclusion	177
19 Connecting to Datomic from Your Application	179
Assumptions	179
Benefits	179
The Recipe—Code	179
Getting Set Up	179
Connecting to Datomic in the Shell	180
Loading Schema and Data	181
Connecting to Datomic from Clojure	181
Connecting to Datomic from Java	182
Connecting to Datomic from a REST Client	185
Conclusion	188
20 Getting Started with Storm	189
Assumptions	189
Benefits	189
The Recipe—Code	190

Testing the Recipe	192
Notes on the Recipe	192
Conclusion	195
21 Getting Started with JMS in Clojure	197
Assumptions	197
Benefits	197
The Recipe—Code	197
Testing the Recipe	200
Notes on the Recipe	200
Conclusion	201
22 Integrating Storm and JMS	203
Assumptions	203
Benefits	203
The Recipe—Code	203
Testing the Recipe	213
Notes on the Recipe	213
Conclusion	215
23 A CSV Reader	217
Assumptions	217
Benefits	217
The Recipe—Code	217
Testing the Solution	219
Notes on the Recipe	219
Conclusion	220
24 Detecting Errors with a Log Monitoring Application	221
Assumptions	221
Benefits	221
The Recipe—Code	221
Testing the Solution	223
Notes on the Recipe	223
Conclusion	224

25	Bundling Clojure as an Ant Plug-in	225
	Assumptions	225
	Benefits	225
	The Recipe—Code	225
	Testing the Recipe	227
	Notes on the Recipe	228
	Conclusion	228
	Postscript—Installing Ant on a Mac	229
26	Bundling Clojure as a Maven Plug-in	231
	Assumptions	231
	Benefits	231
	The Recipe—Code	231
	Testing the Recipe	239
	Notes on the Recipe	240
	Conclusion	240
27	Integrating Clojure by Scripting Web Tests	241
	Assumptions	241
	Benefits	241
	The Recipe—Code	241
	Testing the Recipe	242
	Notes on the Recipe	243
	Conclusion	243
28	Monitoring Availability with a Website Status Checker	245
	Assumptions	245
	Benefits	245
	The Recipe—Code	246
	Testing the Recipe	247
	Notes on the Recipe	249
	Conclusion	249
A	Debugging Macros	251
	Assumptions	251
	Benefits	251

The Recipe	251
A Simple Approach—Expansion-Time and Evaluation-Time stdout	252
Some Macro Helper Functions	253
Read and Evaluate—A More Developed Mental Model	254
Reading	255
Evaluating	257
Conclusion	259

Index	261
--------------	------------

Preface

Who This Book Is For

Clojure Recipes is for people who have started on their journey into Clojure but haven't quite found their feet. Ideally, you are aware that Clojure has lots of parentheses, but the prospect of integrating some libraries to build a working project is still a bit daunting.

If you're comfortable with Clojure and feel like you could easily build a project in a weekend without assistance, then this book is still useful. This is the book you give to someone in the office who is just curious, or who has seen Clojure and wants to get started but needs a helping hand.

Finally, if you're a pragmatist who just needs to get some working code running, then this book is for you. There are lots of examples for you to copy and paste to get your project working.

What This Book Is Not

Clojure Recipes is not an "introduction to Clojure" book. There are some really brilliant books and online resources that cover this topic area. If you want an in-depth explanation of Clojure, then read one of those. This is a "learn by doing" type of book.

What This Book Is About

Clojure Recipes is about "the weekend project." It's about getting something running in a short amount of time. The book makes no assumptions about background knowledge of Clojure, but provides all you need in packaged bites.

The aim of the book is to provide self-contained projects that would have "just enough" for you to get running and see all the pieces hang together. The idea is that you can tweak and extend these projects for your needs.

Why Clojure?

So why should you use Clojure for your next project? Here are a few reasons:

- Clojure focuses on isolating side effects. Whether it is regular business logic or a concurrency scenario, immutability makes your program easier to reason about

and cancels out a whole class of bugs due to state mutation. You can still modify state, but the language encourages you to use it only where necessary.

- Clojure was one of the first languages to make serious use of the ideas in Chris Okasaki's book *Purely Functional Data Structures* (Cambridge University Press, 1999). The big revelation in that book was that data structures implemented in a functional way could be done with an upper bound of time $O(n)$ on costs for reads and writes.
- The benefit of these purely functional data structures was to enable another way to think about Software Transactional Memory (STM) in your program. Clojure introduces constructs like Multiversion Concurrency Control (MVCC) at the application level that previously developers typically only relied upon at the database level.
- Clojure is great at concurrency. In Clojure it is easy to reason about how your program will behave in a concurrency scenario even when there are multiple processes making changes to the one data structure.
- Clojure is pragmatic. Clojure runs on the JVM (in addition to the .NET CLR and JavaScript execution environments). This brings a wealth of libraries from the Java world that can be reused. What's more, when you deploy it, it can look like a jar file so there is no need to tell anyone you're using Clojure at all!
- Clojure has great tooling. Because Clojure runs on the JVM, much of the tooling associated with the JVM for deployment and monitoring is still available to you in the Clojure domain.
- Clojure is a Lisp—one of the oldest programming languages around. This brings a rich heritage of distinctive problem-solving styles and the wisdom of many graybeards who have been chipping away at computing problems for many years.
- Clojure is great at Domain Specific Languages (DSLs). Clojure makes it easy to define languages specifically targeted to the problem you're dealing with. You can easily represent your problem in a completely new way.
- Clojure is fast for developing an end product. Clojure gives you the ability to incrementally compile your program at the REPL, to experiment with it, and to interact with it. With hundreds of functions in the Core libraries, plus access to all the JDK libraries, developers can get a lot done.
- Clojure enables Lean Software Development because, more than other languages, it allows you to delay making decisions about the structure of your program. This comes from a focus on Composition over Inheritance in its idiomatic style, as well as from being able to start writing your program without tying yourself down to a particular expression of the core data.
- Clojure is dynamically typed. Clojure keeps track of the type at run time, so the programmer doesn't have to. This point is enormously controversial at present for three reasons:

1. Types enable you to reason about the behavior of your program in a large system at Compile time. For this reason Clojure enables gradual typing via the `core.typed` library. This way you can add types to your Clojure program as you need it.
 2. Clojure is built to interact with Java, which is typed. The Clojure interop with Java allows type hints and type inference.
 3. Much talk is still being made about Philip Wadler’s papers from the early 1990s on Types and Monads. In particular, proponents claim that Wadler’s ideas enable a compiler-based approach to Edsger Dijkstra’s claim that “[m]uch of the essence of building a program is in fact the debugging of the specification.” Clojure answers this third reason for controversy in two ways. First, Clojure borrows much from the Haskell language and so in a way pays it great homage. Second, the Clojure community values simplicity, probably to the extent that you should know what you want your program to do and whether it is correct, rather than relying on a Compiler to tell you that. (But of course Clojure has a limited ability to use Monads if that’s what you really want.)
- Clojure has the REPL, which is an enormously powerful tool for rapid prototyping, testing, and making changes to your program while it is running.
 - Clojure has uniform syntax (S-expressions), where the primary representation of the program is a data structure in the language itself (sometimes called *homo-ionic*). This might not seem like a big deal, or you might find all the parentheses enormously obnoxious. (Regarding the latter point, several people have suggested that Clojure has fewer parentheses than Java for the equivalent code!) The real benefit of uniform syntax is macros. You can transform and generate Clojure code at Compile time or at run time. One might wonder at the fuss being made over “code as data” until you contrast it with other languages and see the excitement that comes when a new language syntax feature is introduced. Nine times out of ten, that language syntax feature could be implemented as a macro in Clojure. (A key example being `core.async`.)
 - Clojure is highly expressive and extensible. What does the expressiveness of a programming language even refer to? The claim is that you can achieve more with fewer lines of code. Paul Graham made particularly high claims with his Arc challenge. Whether his Arc language was superior was never really answered, but his broader point was that homoionic languages have a natural advantage in being able to express more ideas in fewer lines of code with the power of homoionic syntax and macros.
 - The Clojure community has been enormously innovative. Particular projects of note have been the following:
 - Cascalog—a DSL to generate Hadoop queries with far fewer lines of code.
 - Storm—a highly available, distributed system for processing real-time data.
 - Datomic—a distributed, no-SQL database with point-in-time reproduction of all data, even after “updates.”

- ClojureScript—a JavaScript generator using Clojure syntax enabling DOM transformations and better ways to solve the “callback hell” problem. This is now self-hosting.
- `core.type`—enables “gradual typing” for your application so you can add types as required in order to reason about your program.
- `core.async`—restructures code to provide an inversion of the calling paradigm, allowing the developer to avoid “callback hell,” to better enable the processing of multiple real-time threads of information, and complementing the Clojure implementation with similar syntax and functionality in the browser with ClojureScript.
- Clojure allows lots of different programming styles. Of course, it is idiomatic to write immutable functional code in Clojure. But you could write in a stack-based coding style like FORTH if you wanted. Clojure does have some ability to provide compile-time guarantees via Monads. It is even possible to write procedural blocks of code that mutate state. Clojure is flexible.

So take a look at Clojure! It will be great for your next project.

Coding Conventions Used in This Book

Sometimes we run a command on the command line. It looks like this:

```
lein new myproject
```

Then sometimes we show the result of running a command on the command line. It looks like this:

```
Generating a project called myproject based on the 'default' template.
```

Sometimes we show some code in a Clojure file. It looks like this:

```
(ns myproject)

(defn -main [& args]
  (prn "running"))
```

Sometimes we show a command run on a Clojure REPL. It looks like this:

```
user=> (prn "Hello World")
```

Then sometimes we show the result of running a command on the Clojure REPL. It looks like this:

```
"Hello World"
```

Errata

There is an errata page on the Clojure Recipes website here:
<http://clojurerecipes.net/errata.html>

About the Author

Julian Gamble is a software engineer who has worked in the financial services industry for more than a decade. When he's not enabling billions of dollars to orbit the globe, he writes and presents on all things software related at juliangamble.com/blog. He lives in Sydney, Australia.

This page intentionally left blank

Creating a REST Server in Compojure

In this chapter we will build the first of two REST servers. This is a simple one to do in Compojure. The REST server we build will receive a REST call over http. We will test it using a command line tool, `curl`.

Assumptions

In this chapter we assume the following:

- You have Leiningen installed and on your path on the command line.
- You know how to use `curl` (whether on a Mac or a PC) and have it on your path.
- You understand the concept of http parameters being passed via a GET request URI and via a POST form parameters request.

Benefits

IT organizations now are filled with demands to build backend services implemented in JavaScript for mobile devices like iPhones and rich web clients. Both mobile and rich JavaScript clients work very well with services that implement REST. A great opportunity for you to get Clojure into your organization is by whipping up a REST service in Clojure.

The Recipe—Code

1. Create the project using Leiningen and the Compojure template:

```
lein new compojure rest-demo
```

2. Modify the `project.clj` to have the following contents:

```
(defproject rest-demo "0.1.0-SNAPSHOT"
  :min-lein-version "2.0.0"
  :dependencies [[org.clojure/clojure "1.7.0-beta2"]
                [compojure "1.3.4"]
                [ring/ring-defaults "0.1.5"]]
  :plugins [[lein-ring "0.9.5"]]
  :ring {:handler rest-demo.handler/app}
  :profiles
  {:dev {:dependencies [[javax.servlet/servlet-api "2.5"]
                       [ring/ring-mock "0.2.0"]]}})
```

3. Ensure that the file `rest-demo/src/rest_demo/handler.clj` looks like this:

```
(ns rest-demo.handler
  (:require [compojure.core :refer :all]
            [compojure.route :as route]
            [ring.middleware.defaults
             :refer [wrap-defaults site-defaults]]))

(defn handle-http []
  (context "/:id" [id]
    (defroutes api-routes
      (GET "/" [] (str "get called: " id "\n"))
      (POST "/" {form-params :form-params}
        (str "post called: " id "\n" form-params " \n"))
      (PUT "/" req (str "put called with params: " req))
      (DELETE "/" [] (str "delete called: " id "\n"))))

  (defroutes app-routes
    (handle-http)
    (route/not-found (str
      "This is the default page - try "
      "<a href='http://localhost:4000/33'>this</a>\n")))

  (def app
    (wrap-defaults app-routes
      (assoc-in site-defaults [:security :anti-forgery] false)))
```

Testing the Solution

Let's give it a run.

1. In the command prompt, change back to the parent directory of your project and start the server using Leiningen:

```
lein ring server-headless 4000
```

Note that we are using the `server-headless` parameter to the `lein ring` command. This starts the server without opening a web browser. Had we merely run (as a hypothetical) `lein ring server 4000`, a new web browser would have opened. (But because we're about to use `curl` to interact with the website, a browser would have gotten in the way.)

2. Open a new command window and type:

```
curl http://localhost:4000/1
```

You should get a response like:

```
get called: 1
```

3. Now enter:

```
curl -X DELETE http://localhost:4000/4
```

You should get a response like:

```
delete called: 4
```

4. Now enter:

```
curl -X POST -d "id=2" http://localhost:4000/3
```

You should get a response like:

```
post called: 3
{"id" "2"}
```

5. Now enter:

```
curl -X PUT -d "id=2" http://localhost:4000/3
```

You should now get something similar to:

```
put called with params: {:ssl-client-cert nil, :remote-addr
"0:0:0:0:0:0:1%0", :scheme :http, :query-params {}, :context "/3",
:form-params {"id" "2"}, :request-method :put, :query-string nil,
:route-params {:id "3"}, :content-type "application/x-www-form-urlencoded",
:path-info "/", :uri "/3", :server-name "localhost", :params {:id "3", "id"
"2"}, :headers {"user-agent" "curl/7.27.0", "content-type" "application/
x-www-form-urlencoded", "content-length" "4", "accept" "**/*", "host"
"localhost:4000"}, :content-length 4, :server-port 4000, :character-encoding
nil, :body #<HttpInput org.eclipse.jetty.server.HttpInput@cfefc0>}
Julians-MacBook-Pro:~
```

You can see a large amount of information is in the parameter map `req`. This also demonstrates the powerful, dynamic, interactive nature of running Compojure with the Leiningen plug-in. You can modify the file and save it, and your changes are accessible instantly to the web browser. (We could also have done all of this on the REPL—but we'll save that for another day.)

Notes on the Recipe

The Compojure library is designed to make RESTful URIs easy to work with. Note in particular the `project.clj` file:

```
[ring/ring-defaults "0.1.5"]
:plugins [[lein-ring "0.9.5"]]
:ring {:handler rest-demo.handler/app}
```

Notice the `ring-defaults` library. We'll use this when we examine parameters passed in. Also note the `lein-ring` plug-in. This enables us to start the app from the command line. It will also enable us to modify the app when it is running and to see the results without restarting the server.

Also note the `:ring {:handler . . .}` syntax. This points to the part of the application that will handle the incoming requests.

Now look at the file `handler.clj`, in particular the namespace:

```
(ns rest-demo.handler
  (:require [compojure.core :refer :all]
            [compojure.route :as route]
            [ring.middleware.defaults
             :refer [wrap-defaults site-defaults]]))
```

Note that we load the Compojure libraries for handling routes, and we use the `ring.middleware` for parameter handling.

Now observe the `handle-http` function definition:

```
(defn handle-http []
  (context "/" :id [id]
    (defroutes api-routes
      (GET "/" [] (str "get called: " id "\n"))
      (POST "/" {form-params :form-params}
        (str "post called: " id "\n" form-params " \n"))
      (PUT "/" req (str "put called with params: " req))
      (DELETE "/" [] (str "delete called: " id "\n")))))
```

This function handles the parameters for the particular http request type. Here we just do simple handlers for the different http request types. This handles requests in the format:

```
http://servername/2
```

Now note the following function:

```
(defroutes app-routes
  (handle-http)
  (route/not-found (str
    "This is the default page - try "
    "<a href='http://localhost:4000/33'>this</a>\n")))
```

This is the main route-handler function. It delegates most of its responsibilities to our `handle-http` function above. If that returns `nil`, then it displays a "Not Found" response.

Now note the following symbol:

```
(def app
  (wrap-defaults app-routes
    (assoc-in site-defaults [:security :anti-forgery] false)))
```

This is the application hook. We point to this in the `project.clj` file. It takes the route definitions in `app-routes` and feeds that into the function result of `wrap-defaults`. The `wrap-defaults` function adds middleware to the route to enable URI parameter input. We switch off the `anti-forgery` middleware so our simple `curl` tests will work. You shouldn't do this in your production application.

Conclusion

We've seen an example of generating RESTful HTTP requests from the command line using the `curl` command, and we implemented handlers for these requests in `Compojure`.

This page intentionally left blank

Index

A

- Ant
 - downloading, 228
 - installing on a Mac, 229
- Ant plug-in, 225–229
 - assumptions before beginning, 225
 - benefits of, 225
 - notes on the recipe for, 228
 - recipe—code for, 225–227
 - testing the recipe for, 227–228
- Aspect Orientation, in Object-Oriented languages, 76–77

B

- Big data, 139
- Blocking code, macro converting to non-blocking code, 79–85
- Bolts, in Storm
 - creating, 190–192
 - description of, 191, 192–193, 194–195
 - integrating Storm and JMS using, 204, 205, 206, 208–209, 213–214, 215
- Bootstrap, 56, 59, 62

C

- Cascading library in Hadoop
 - introduction of, 141
 - Java class dependency relationships with, 177
 - preformatted data needed for, 160–161
 - reading files in, 174–175, 177
- Cascalog, introduction to, xvii, 135–141
 - background concepts for, 140–141
 - benefits of Cascalog, 135
 - notes on the recipe for running basic queries in, 137–141
 - recipe—code for running basic queries in, 135–137
 - testing the solution in, 137–139
- Cascalog, compiling to a jar file and running, 143–148
 - assumptions before beginning, 143
 - benefits of, 143
 - recipe—code for, 143–145
 - testing the solution in, 145–146
- Cascalog, loading custom data formats into, 163–177
 - assumptions before beginning, 163
 - benefits of, 163
 - notes on the solution for, 176–177
 - recipe—code for, 163–176
 - testing the recipe for, 175–176
- Cascalog, loading data file into, 149–151
 - assumptions before beginning, 149
 - benefits of, 149
 - recipe—code for, 149–150
 - testing the solution in, 150–151
- Cascalog, loading structured data into, 157–161
 - assumptions before beginning, 157
 - benefits of, 157
 - notes on solution for, 160–161
 - preformatting data for, 160–161
 - recipe—code for, 157–159
 - testing the recipe for, 159–160
- Cascalog, structuring a file to load into, 149–151
 - assumptions before beginning, 149
 - benefits of, 149
 - recipe—code for, 149–150
 - testing the solution in, 150–151
- Cascalog, writing out a data file with, 153–155
 - assumptions before beginning, 153
 - benefits of, 153
 - notes on the recipe for, 155
 - recipe—code for, 153–154
 - testing the solution in, 154–155
- Clojure, reasons for using, xv–xvii
- clojure.csv library, 219
- clojure.data.json library, in Compojure, 30
- ClojureScript, xvii
 - compiled to JavaScript, 35, 39–40
 - JSON server built in, 41–46
 - Pedestal server instead of, 50
- ClojureScript REST client, 35–40
 - assumptions before beginning, 35
 - benefits of, 35
 - notes on the recipe for, 39–40
 - recipe—code for, 36–38
 - testing the solution in, 38–39
- ClojureWerkz money library, 70, 74
- Coding conventions in text, xviii
- Comma-separated value (CSV) files. *See* CSV reader
- Compiler extension using macros, 79–85
 - assumptions before beginning, 79
 - benefits of, 79
 - notes on the recipe for, 82–85
 - recipe—code for, 79–81
 - testing the solution in, 81–82

Conventions in text, xviii
 core.async library, 85
 CSS in Clojure, 107–133
 mapping of SASS constants into CSS data structure using, 129–133
 process of going from SASS to CSS in, 108–109
 SASS for reuse and manageability of, 107–108
 transformation of tree representation of, 126–128
 CSV reader, 217–220
 assumptions before beginning, 217
 benefits of, 217
 notes on the recipe for, 219–220
 recipe—code for, 217–218
 testing the solution for, 219
 curl, and REST servers, 20–21, 23, 25, 29, 179, 187

D

data.csv library, 219
 data.json library, 29, 30
 Datalog, 138, 141
 Datomic, xvii
 downloading, 180
 setting up, 179–180
 Datomic connections, 179–188
 assumptions before beginning, 179
 benefits of, 179
 from Clojure, 181–182
 getting set up for, 179–180
 from Java, 182–185
 loading schema and data in, 181
 recipe—code for, 179–180
 from REST client, 185–188
 in the shell, 180–181
 Datomic syntax, and DSL, 87–106
 assumptions before beginning, 87
 benefits of, 87–88
 notes on the recipe for, 103–106
 recipe—code for, 88–97
 relational database mindset applied to, 88
 testing add datom in, 98–100
 testing add nested datom in, 101–104
 testing create nested schema in, 100–101
 testing create schema in, 97–98
 Debugging macros, 251–259
 assumptions before beginning, 251
 benefits of, 251
 helper functions and, 253–254
 logging application with, 71–72, 73, 74–75
 read and evaluate approach for, 254–259
 simple approach for, 252–253
 Dijkstra, Edsger, xvi
 DSL (domain-specific language)
 Clojure with, xvi
 Datomic syntax and, 87–106
 SASS and, 107–133

E

EDN, 47
 Error log monitoring application, 221–224
 assumptions before beginning, 221
 benefits of, 221
 notes on the recipe for, 223–224
 recipe—code for, 221–223
 testing the solution for, 223
 EventSource, 55, 60–61, 63, 65–66
 Excel, and CSV files, 217, 219, 220

F

Fielding, Roy, 33
 Firefox, 241, 243
 future, with compiler extension macro, 81, 82, 83–84

G

Google Closure libraries, 35, 39
 Google MapReduce, 139

H

Hadoop
 as a batch processing system, 149
 Cascading library in, 141
 downloading, 146, 147
 MapReduce concept and, 140
 Pig analysis tool with, 140
 setting up on a Mac, 146–147
 setting up on Windows, 147–148
 handle-http function, 22–23
 Haskell language, xvi
 Hickey, Rich, 26, 50
 Homoiconic language, xvii, 69
 Hornet libraries, with JMS, 197, 198, 200, 213, 214–215
 HTML5, and EventSource, 55, 60–61, 63, 65–66
 http requests
 JSON server and, 41–47
 Pedestal stock ticker application using, 55–67
 REST server and, 22–23
 http specification, and REST, 32–33

I

Interceptors, 51, 54

J

Java
 checking for on a Mac, 3
 checking for on Windows, 2
 Clojure macro use and, 76–77
 Datomic connection from, 182–185
 Swing library in, 240
 Java Development Kit. *See* JDK

- Java EE (Java Platform, Enterprise Edition), 11–18
 - assumptions before beginning, 11
 - benefits of, 11
 - packaging Clojure to work in a Java EE environment and, 11–12
 - recipe (common part) for, 11–14
 - setting up Tomcat on a Mac and, 14–15
 - setting up Tomcat on Windows and, 16–18
- Java Platform, Enterprise Edition. *See* Java EE
- javac, checking for, on a Mac, 4
- JavaScript, xv, 35, 39–40
- JDK (Java Development Kit), xvi
 - downloading, 2, 3
 - setting up Hadoop on a Windows machine and, 148
 - setting up on a Mac, 4–5
 - setting up on Windows, 5–8
- JMS feeds, and Storm, 203–214
 - assumptions before beginning, 203
 - benefits of, 203
 - notes on the recipe for, 213–215
 - recipe—code for, 203–213
 - testing the recipe for, 213
- JMS server, setting up and messaging, 197–201
 - assumptions before beginning, 197
 - benefits of, 197
 - notes on the recipe for, 200–201
 - recipe—code for, 197–200
 - testing the recipe for, 200
- JQuery, 41, 46, 55, 62
- JRE (Java Runtime Environment), checking for and adding, on a Mac, 4–5
- JSON server, 41–46
 - assumptions before beginning, 41
 - benefits of, 41
 - notes on the recipe for, 45–46
 - recipe—code for, 41–44
 - testing the solution in, 44–45
 - Transit used with, 47
- JVM, xv
- K**
- Kay, Alan, 69
- L**
- Lean Software Development, xvi
- lein-ring library, 45
- Leiningen
 - assumptions before beginning, 1
 - benefits of, 1
 - creating new project in projects directory (Mac) with, 2–3
 - creating new project in projects directory (Windows) with, 4
 - description of, 1
 - new project directory in, 11
 - Pedestal stock ticker application using, 55, 56, 60
 - Pedestal web server using, 50, 52
 - recipe (Mac) for, 3–4
 - recipe (Windows) for, 1–3
 - starting a project with, 1–9
- Liberator REST server, 25–33
 - assumptions before beginning, 25
 - benefits of, 25
 - context of http specification for, 32–33
 - notes on the recipe for, 29–32
 - recipe—code for, 25–27
 - testing the solution in, 28–29
- Liberator library, in Compojure, 29, 30–31, 32
- Lisp, xv
- Log monitoring application, 221–224
 - assumptions before beginning, 221
 - benefits of, 221
 - notes on the recipe for, 223–224
 - recipe—code for, 221–224
 - testing the solution for, 223
- Logging application macro, 69–77
 - assumptions before beginning, 69
 - benefits of, 69
 - debug macros in, 71–72, 73, 74–75
 - notes on the recipe for, 74–77
 - recipe—code for, 70–72
 - testing the solution in, 72–73
- M**
- Macro compiler extension, 79–85
 - assumptions before beginning, 79
 - benefits of, 79
 - notes on the recipe for, 82–85
 - recipe—code for, 79–81
 - testing the solution in, 81–82
- Macro debugging, 251–259
 - assumptions before beginning, 251
 - benefits of, 251
 - helper functions and, 253–254
 - logging application with, 71–72, 73, 74–75
 - read and evaluate approach for, 254–259
 - simple approach for, 252–253
- Macro logging application, 69–77
 - assumptions before beginning, 69
 - benefits of, 69
 - debug macros in, 71–72, 73, 74–75
 - notes on the recipe for, 74–77
 - recipe—code for, 70–72
 - testing the solution in, 72–73
- Macros
 - trace¹, 77
 - uses of, in Clojure, 76–77
- MapReduce
 - Google introduction of, 139
 - Yahoo’s MapReduce and, 140

Marz, Nathan, 141, 192

Maven, 182–183

- debugging a Leiningen project as, 164
- Datomic connection from Java and, 182–185
- Leiningen setup and, 1, 4

Maven plug-in, 231–240

- assumptions before beginning, 231
- benefits of, 231
- notes on the recipe for, 240
- recipe—code for, 231–239
- testing the recipe for, 239

Middleware

- in Ring, 54
- using interceptors instead of, 54
- wrap-default function for, 23, 39

Monads, xvii

money library, ClojureWerkz, 70, 74

Monitoring applications

- log monitoring application, 221–224
- website status checker, 245–249

Multiversion Concurrency Control (MVCC), xv

N

.NET CLR, xv

O

Object-Oriented languages, and Aspect Orientation, 76–77

Okasaki, Chris, xv

P

Pedestal simple server, 49–54

- assumptions before beginning, 49
- benefits of, 49
- context for, 49–50
- notes on the recipe for, 53–54
- reasons for using instead of Compojure, 50
- recipe—code for, 50–52
- testing the solution in, 52–53

Pedestal stock ticker application, 55–67

- assumptions before beginning, 55
- benefits of, 55
- notes on the recipe for, 62–67
- recipe—code for, 55–60
- testing the solution in, 60–62

Pig analysis tool, with Hadoop, 140

Production applications

- Clojure as Ant plug-in in, 225–229
- log monitoring with, 221–224
- WebDriver tests with, 241–243
- website status checker with, 245–249

R

Reducers, 139–140

Regex, 223–224

REPL (Read Eval Print Loop), xvi, 4, 137

REST (Representational State Transfer)

- http specification and, 32–33
- origins of, 33
- Richardson Maturity Model for, 33

REST clients

- connecting to Datomic from, 185–188
- documentation for, 185

REST server in Compojure, 19–23

- assumptions before beginning, 19
- benefits of, 19
- http requests with, 22–23
- notes on the recipe for, 22–23
- recipe—code for, 19–20
- testing the solution in, 20–21

REST server in Liberator, 25–33

- assumptions before beginning, 25
- benefits of, 25
- http specification and, 32–33
- notes on the recipe for, 29–32
- recipe—code for, 25–27
- testing the solution in, 28–29

RESTful HTTP requests, 22–23

Richardson Maturity Model for REST, 33

ring.json library, 45

Ruby, 107, 133

S

S-expressions, xvi–xvii

SASS

- constants in, 108
- features of, 107–108
- larger process of going from SASS to CSS in, 108–109
- mapping of constants into data structure using, 129–133
- nesting in, 108, 125–126
- reuse and manageability of CSS using, 107–108
- transformation of tree representation in, 126–128

SASS DSL with Clojure zippers, 107–133

- assumptions before beginning, 107
- benefits of, 107
- features of SASS and, 107–108
- mapping of constants into data structure using, 129–133
- notes on the recipe for, 122–133
- recipe—code for, 109–121
- tasks accomplished in, 109
- testing the solution in, 121–122
- transformation of tree representation in, 126–128

Selenium WebDriver tests, 241–243

- assumptions before beginning, 241
- benefits of, 241
- notes on the recipe for, 243

- recipe—code for, 241–242
 - testing the recipe for, 242–243
- Servers. *See* JMS server; JSON server; Pedestal simple server; REST server in Compojure; REST server in Liberator
- Software Transactional Memory (STM), xv
- Spouts, in Storm
 - creating, 190–192
 - description of, 192–193, 195
 - integrating Storm and JMS using, 203, 208–209, 214, 215
- Stock ticker application, Pedestal, 55–67
 - assumptions before beginning, 55
 - benefits of, 55
 - notes on the recipe for, 62–67
 - recipe—code for, 55–60
 - testing the solution in, 60–62
- Storm, and JMS feeds, 203–214
 - assumptions before beginning, 203
 - benefits of, 203
 - notes on the recipe for, 213–215
 - recipe—code for, 203–213
 - testing the recipe for, 213
- Storm apps, xvii
 - components of, 192–193
 - getting started with, 189–190
 - assumptions before beginning, 189
 - benefits of, 189
 - notes on the recipe for, 192–195
 - recipe—code for, 190–192
 - testing the recipe in, 192
- Swing
 - with Ant, 226, 228
 - with Maven, 232, 240

T

- Test suites, 241–243
- Tomcat on a Mac
 - assumptions before setting up, 14
 - downloading, 14
 - notes on installing, 14
 - packaging Clojure to work in a Java EE environment and, 11–12
 - setting up, 14–15
- Tomcat on Windows
 - assumptions before setting up, 16
 - downloading, 16

- packaging Clojure to work in a Java EE environment and, 11–12
 - setting up, 16–18
- tools.trace library, 77
- Topologies, in Storm
 - description of, 192, 193, 195
 - integrating Storm and JMS using, 203, 204–205, 206, 208–209, 214, 215
- trace¹ macro, 77
- Transit, 47
- Tuples, in Storm
 - description of, 192, 193, 194–195
 - integrating Storm and JMS using, 205–206, 214
- Twitter
 - Backtype, 141n, 192
 - Bootstrap and, 56, 59, 62
- Types, xvii

W

- Wadler, Philip, xvi
- WebDriver tests, 241–243
 - assumptions before beginning, 241
 - benefits of, 241
 - notes on the recipe for, 243
 - recipe—code for, 241–242
 - testing the recipe for, 242–243
- Website status checker, 245–249
 - assumptions before beginning, 245
 - benefits of, 245
 - notes on the recipe for, 249
 - recipe—code for, 246–247
 - testing the recipe for, 247–249
- WebSockets, 55
- wget.exe, 2
- wrap—default function, 23

Y

- Yahoo, 140

Z

- Zippers
 - custom zipper for maps in Datomic syntax, 91–94, 104
 - notes on the recipe for SASS DSL on using, 122–133
 - parsing a DSL using, 107
 - recipe—code for generating CSS with, 109–121
 - traverse idiom for, 123, 126

This page intentionally left blank

PEARSON

InformIT is a brand of Pearson and the online presence for the world's leading technology publishers. It's your source for reliable and qualified content and knowledge, providing access to the leading brands, authors, and contributors from the tech community.

▼ Addison-Wesley **Cisco Press** **IBM Press** Microsoft Press

PEARSON
IT CERTIFICATION

PRENTICE
HALL

QUE

SAMS

vmware PRESS

LearnIT at InformIT

Looking for a book, eBook, or training video on a new technology? Seeking timely and relevant information and tutorials. Looking for expert opinions, advice, and tips? **InformIT has a solution.**

- Learn about new releases and special promotions by subscribing to a wide variety of monthly newsletters. Visit informit.com/newsletters.
- FREE Podcasts from experts at informit.com/podcasts.
- Read the latest author articles and sample chapters at informit.com/articles.
- Access thousands of books and videos in the Safari Books Online digital library. safari.informit.com.
- Get Advice and tips from expert blogs at informit.com/blogs.

Visit informit.com to find out all the ways you can access the hottest technology content.

Are you part of the **IT** crowd?

Connect with Pearson authors and editors via RSS feeds, Facebook, Twitter, YouTube and more! Visit informit.com/socialconnect.



REGISTER



THIS PRODUCT

informit.com/register

Register the Addison-Wesley, Exam Cram, Prentice Hall, Que, and Sams products you own to unlock great benefits.

To begin the registration process, simply go to **informit.com/register** to sign in or create an account.

You will then be prompted to enter the 10- or 13-digit ISBN that appears on the back cover of your product.

Registering your products can unlock the following benefits:

- Access to supplemental content, including bonus chapters, source code, or project files.
- A coupon to be used on your next purchase.

Registration benefits vary by product. Benefits will be listed on your Account page under Registered Products.

About InformIT — THE TRUSTED TECHNOLOGY LEARNING SOURCE

INFORMIT IS HOME TO THE LEADING TECHNOLOGY PUBLISHING IMPRINTS Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que, and Sams. Here you will gain access to quality and trusted content and resources from the authors, creators, innovators, and leaders of technology. Whether you're looking for a book on a new technology, a helpful article, timely newsletters, or access to the Safari Books Online digital library, InformIT has a solution for you.

informIT.com

THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison-Wesley | Cisco Press | Exam Cram
IBM Press | Que | Prentice Hall | Sams

SAFARI BOOKS ONLINE