Incanter

A User's Guide to Data Sorcery

Notice

Notice

Topics:

This information was developed for products and services offered in the U.S.A.

This product is meant for educational purposes only. Some of the trademarks mentioned in this product appear for identification purposes only. Not responsible for direct, indirect, incidental or consequential damages resulting from any defect, error or failure to perform. Any resemblance to real persons, living or dead is purely coincidental. Void where prohibited. Some assembly required. Batteries not included. Use only as directed. Do not use while operating a motor vehicle or heavy equipment. Do not fold, spindle or mutilate. Do not stamp. No user-serviceable parts inside. Subject to change without notice. Drop in any mailbox. No postage necessary if mailed in the United States. Postage will be paid by addressee. Post office will not deliver without postage. Some equipment shown is optional. Objects in mirror may be closer than they appear. Not recommended for children. Your mileage may vary.

No other warranty expressed or implied. This supersedes all previous notices.

COPYRIGHT LICENSE:

Trademarks

The following terms are trademarks of the Foobar Corp. in the United States, other countries, or both:

Foobarista ®

The following terms are trademarks of other companies:

Red, Orange, Yellow, Green, Blue, Indigo, and Violot are registered trademarks of Rainbow Corporation and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Contents

Abstract

User Guide for Incanter. See also the Technical Manual.

Preface

About This Work

Plan of the Work

The basic approach: describe the problem, then describe general strategies for addressing the problem, then describe Incanter-specific solutions, then give an example.

Key distinctions: Incanter “task” (i.e. command) v. developer's task (e.g. test, document); project map (i.e. configuration) v. command

Note: Incanter's terminology should be changed to use “command” where it currently uses “task”. This is because the tasks are not in fact tasks; e.g. `lein compile` is a command from the user to Incanter rather than a task to be accomplished. We need to reserve the word “task” to refer to “user tasks”, the tasks confronting the developer, which appear as problems to be solved. So for example the first task is merely to configure the project. The way to solve this problem is to write the project.clj and set up the directory structure.

Plan of the work:

Overview	installation, projects, infrastructure
Project Maps	How the projmap is constructed at runtime; how tasks use the projmap to drive configuration of the
Tasks	Two kinds of task: • Dev tasks, e.g coding, testing, QA, etc • Incanter “tasks”, i.e. commands General description of tasks; commonly used tasks with examples
Infrastructure: Respositories	role of repos; how to specify, etc. and how Incanter searches, etc
etc	etc

Chapter 1

What is Incanter?

Topics:

Incanter is a collection of commands that address developer tasks and needs. It depends on a kernel of core functionality, a set of plugins that implement the commands available to users, and a set of external resources - directories, files, maven repositories, etc.

Incanter can be compared to Emacs. Emacs consists of a kernel of core functionality (implemented in C) made available to the user by means of commands or functions defined in Elisp, just like user-defined functions and commands. Incanter also has a kernel of core functionality (implemented in Clojure) that is made available to users by means of *plugins*, which define Incanter *commands*; user-defined commands are implemented in the same way.

Structure & Dynamics of Incanter Projects

Structure of project: project config map (in project.clj) determines dir structure.

Dynamics: when a lein command is invoked, lein dynamically constructs the effective project map and invokes the command, passing the EPM as arg. The command inspects the map and uses its content to control its processing.

Project Maps

The central concept of Incanter is the *project map*.

What does a project map look like? An ordinary Clojure map: set of key-val pairs. In the project map, keys are always Clojure :-prefixed keys. L defines a default set of keys; users are free to extend this set.

The Project Configuration Map

The *project configuration map* is defined by `defproject` macro in the `project.clj` file in the project *root directory*.

Terminology: the *project configuration map* is specified as part of `defproject` in `project.clj`. The *effective project map* is dynamically constructed by combining the project config map and various other maps - see X for details.

Profiles

Profiles provide a means of customizing the effective project map.

A profile is a named map. Incanter predefines several *system profiles*: :base, :system, :user, :dev, and :provided. The system profiles are always in effect; user-defined can override and/or extend them. User-defined profiles can be specified in several places:

At runtime, Incanter integrates profiles with the project configuration map of `project.clj` to produce the effective project map.

The Effective Project Map

The *effective project map* is dynamically constructed by combining the project config map and various other maps

Incanter command implementations receive the effective project map as an argument when they are invoked.

General description of how L constructs the *effective project map* from project.clj/defproject in combination with various other maps. This is one of the basic jobs of the kernel. For details, refer to the Project Map node.

Project Map Semantics

Project map semantics are determined by command implementations.

What are the semantics of the (effective) project map? Determined entirely by the command implementations. All Incanter commands use the effective project map, which is delivered by Incanter to the command as its sole argument. Different commands pick out different parameters from the map for use in controlling processes.

Commands

Incanter *commands* (formerly “tasks”) expose functionality to the user.

Commands are what you would expect: something you type to tell lein what to do.

L comes with a set of predefined commands. Users can extend this set by writing a *plugin*, which is the implementation of a command. There is no technical difference between L's predefined commands and user-defined commands.

The Incanter Infrastructure

Incanter's infrastructure config files, the local repo, remote repos, etc. We should also include default profiles such as :base, since they are just as fundamental. This topic provides a brief overview of these parts and how they fit together.

External

Directories, files, env vars used by Incanter

Directories: ~/.lein, ~/lein/profiles.d, etc; ~/.m2/repository;

Files: ~/.lein/profiles.clj

Environment variables

Command line options

Anything else Incanter can take from the env?

Do this

Internal infrastructure: profiles, etc.

Profiles and other internally defined (but overridable) stuff on which Incanter's functionality depends.

Getting Started

We get started by first exploring a minimal project, and then exploring the Incanter infrastructure. This should give us a good general idea of what pieces are involved and how the work together.

Before delving into the details we take a brief tour of:

- A minimal project
- The Incanter infrastructure

Installation

Unix-like systems

If your preferred *package manager* has a relatively recent version of Incanter, try that first. Otherwise you can install by hand:

Incanter bootstraps itself using the `lein` shell script; there is no separate install script. It handles installing its own dependencies, which means the first run will take longer.

1. Make sure you have a Java JDK version 6 or later.
2. *Download the script*
3. Place it on your *$PATH*. (`~/bin` is a good choice if it is on your path.)
4. Set it to be executable. (`chmod 755 ~/bin/lein`)

Windows

There is an *installer* which will handle downloading and placing Incanter and its dependencies.

The manual method of putting the *batch file*. on your *PATH* and running `lein self-install` should still work for most users. If you have Cygwin you should be able to use the shell script above rather than the batch file.

A Quick Tour

We create a minimal project and explore its structure and configuration.

A minimal Incanter project involves: a directory structure, a project configuration file (project.clj), and source code files. In addition to these static resources, Incanter dynamically determines a project map, a dependency tree, and several other structures. This section takes you through some simple steps to exlore these elements.

1. Create a new project by executing: `lein new app my-app`

   where `lein` is the Incanter command, and the args are:
   new

a Incanter *task*. To see the syntax and semantics of this task, run `lein help new`. To see a list of the tasks that come preinstalled, run `lein help`.

app

is the name of a project template, in this case the default application template. To see a list of preinstalled templates, run `lein help new`. Incanter's concept of *task* is discussed in detail under the X topic of this guide.

my-app

is the name to be used for the new project

This will create a hierarchy of directories and populate it with some files generated from templates:

```
.
./.gitignore
./doc
./doc/intro.md
./project.clj
./README.md
./src
./src/my_app
./src/my_app/core.clj
./test
./test/my_app
./test/my_app/core_test.clj
```

2. Take a look at the Incanter project file buy running `less my-app/project.clj`
You should see something like:

```
(defproject my-app "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.5.1"]])
```

Note: This has the form of a function application, where `defproject` is the function, and the args come as a list of pairs. In fact, `defproject` is a macro that expands into the definition of a Clojure map named `project`. Each task - both those preinstalled and those defined by plugins - takes this `project` map as input. The key point is that the key-value pairs of the `project` map are thus available for use by task implementations to control setting of options etc. for the processes they manage.

Note: The final `project` map is determined by a combination of several sources in addition to project.clj, such as *~/.lein/profiles.clj*. See X for details.

In this minimal example, the only really functional parameter is `:dependencies`. This is used to specify which libraries/jars the project uses (and thus depends on). One of the best things about Incanter is its powerful management of dependencies. Using the [?] library under the hood, Incanter is able to construct the entire dependency graph for the project and arrange for everything needed to be installed. Incanter's dependency management capabilities are described in X.

Important: Incanter dependency specification uses the naming conventions established by Maven: artifact-group/artifact-id. Etc. See X for details.

3. Examine the project map: `lein pprint`
You should see something like:

```
{:compile-path "/Users/gar/tmp/my-app/target/classes",
 :group "my-app",
 :license
```

```
 {:name "Eclipse Public License",
  :url "http://www.eclipse.org/legal/epl-v10.html"},
 :global-vars {},
 :checkout-deps-shares
 [:source-paths
  :test-paths
  :resource-paths
  :compile-path
  #&Var@3e25e2b8:
    #amp;classpath$checkout_deps_paths leiningen.core.classpath
$checkout_deps_paths@89bbe8c>>],
 :dependencies
 ([org.clojure/clojure "1.5.1"]
  [org.clojure/tools.nrepl "0.2.3" :exclusions ([org.clojure/clojure])]
  [clojure-complete/clojure-complete
   "0.2.3"
   :exclusions
   ([org.clojure/clojure])]),
 :plugin-repositories
 [["central" {:snapshots false, :url "http://repo1.maven.org/maven2/"}]
  ["clojars" {:url "https://clojars.org/repo/"}]],
 :test-selectors {:default (constantly true)},
 :target-path "/Users/gar/tmp/my-app/target",
 :name "my-app",
 :deploy-repositories
 [["clojars"
   {:username :gpg,
    :url "https://clojars.org/repo/",
    :password :gpg}]],
 :root "/Users/gar/tmp/my-app",
 :offline? false,
 :source-paths ("/Users/gar/tmp/my-app/src"),
 :certificates ["clojars.pem"],
 :version "0.1.0-SNAPSHOT",
 :jar-exclusions [#"^\."],
 :profiles {:uberjar {:aot :all}},
 :prep-tasks ["javac" "compile"],
 :url "http://example.com/FIXME",
 :repositories
 [["central" {:snapshots false, :url "http://repo1.maven.org/maven2/"}]
  ["clojars" {:url "https://clojars.org/repo/"}]],
 :resource-paths
 ("/Users/gar/tmp/my-app/dev-resources"
  "/Users/gar/tmp/my-app/resources"),
 :uberjar-exclusions [#"(?i)^META-INF/[^/]*\.(SF|RSA|DSA)$"],
 :main my-app.core,
 :jvm-opts ["-XX:+TieredCompilation" "-XX:TieredStopAtLevel=1"],
 :eval-in :subprocess,
 :plugins
 ([lein-localrepo/lein-localrepo "0.4.1"]
  [lein-difftest/lein-difftest "1.3.8"]
  [org.clojure/java.classpath "0.2.0"]
  [lein-marginalia/lein-marginalia "0.7.1"]
  [lein-mustache/lein-mustache "0.2"]
  [lein-pprint/lein-pprint "1.1.1"]),
 :native-path "/Users/gar/tmp/my-app/target/native",
 :description "FIXME: write description",
 :test-paths ("/Users/gar/tmp/my-app/test"),
 :clean-targets [:target-path],
 :aliases nil}
```

Take some time to look this over. Most of these parameters you will never have to deal with, but it's good to have an idea of what sorts of things Incanter is interested in.

Note: The predefined paramters are documented in the Technical Reference Manual. Other chapters of this User's Guide explain how to use them.

Note: We should distinguish between the project map that results from Incanter's work and the defproject map in project.clj that forms the starting point for project map construction.

4. Tell Incanter to install all dependencies by running `lein deps`

   Since you already have Clojure installed, you probably won't see any output.

5. Check your dependency tree: `lein deps :tree`
   You should see something like the following:

```
[clojure-complete "0.2.3" :exclusions [[org.clojure/clojure]]]
[org.clojure/clojure "1.5.1"]
[org.clojure/tools.nrepl "0.2.3" :exclusions [[org.clojure/clojure]]]
```

   This is a complete listing of the jars your project depends on, derived from your project.clj :dependencies parameter plus a set of default maps to be described later. Since this is a tree representation, you can infer that these three libraries are independently specified; none of them has required any of the others as a dependency. In fact, `clojure-complete` and `org.clojure/tools.nrepl` are both specified as dependencies by the default `:base` *profile*, which means that they will always be in the dependency tree for every project (unless overriden). Profiles are named maps that can be used to customize the `project` map in various ways; they are fully described in X.

6. Now let's take a look at the application code installed by the `app` template. Use your editor, or run `cat src/my_app/core.clj`

   The contents of `my-app/src/my_app/core.clj` should look something like:

```
(ns my-app.core
  (:gen-class))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))
```

Notice: The `:gen-class` option and the definition of `-main`. Remember this was produced by the application template, rather than the library template. So it assumes you want to execute the result, which means you need to generate Java byte code. That's what :gen-class does. You also need a main entry point; thats what `-main` does.

Warning: In case it isn't obvious: to effectively use Incanter, you need to know Clojure. That is, the better your mastery of Clojure, the more you can do with Incanter. Incanter is a Clojure application, after all.

Part I

Using Incanter

Topics:

Chapter 1

Working with Data

Topics:

Importing Data

Part II

Extending Incanter

Topics:

Chapter 1

Template Development

How to implement custom templates.

Should this go in a separate “Developer's Manual”?

Chapter 2

Plugin Development

How to implement custom tasks (plugins).

Should this go in a separate “Developer's Manual”?

Chapter 3

Profile Development

How to implement custom profiles.

Should this go in a separate “Developer's Manual”?

Chapter 4

Embedding Incanter

Incanter is a Clojure library. You can use it from your own code.

Should this go in a separate “Developer's Manual”?

Appendix

Appendices

Topics:

Appendix
A

Contributing to this document

WARNING: Document design can be hazardous to your other interests. (Apologies to Knuth)

This is a DITA document. The original is available from the *github project*.

Please use the *Issues Tracker* to register bugs, corrections, enhancement requests, etc. Alternatively, you can clone the repository and edit the text directly. If you do please credit yourself in the metadata.

Intellectual property

DITA

DITA (Darwin Information Typing Architecture) is an *OASIS Standard* for writing, managing, and publishing information. It supports "topic-based" authoring.

The *DITA Wiki Knowledgebase* contains lots of information on DITA, from introductions and overviews to detailed documentation.

Another good overview is *DITA for the Impatient* from *XMLMind*.

The *DITA Language Specification* documents the elements and atributes of the DITA Schema.

Tools

There are two open source DITA transformation tools available:

- *DITA Open Toolkit*
- *XMLMind DITA Converter*