How to Make a LightTable

David Siegel

DesignTable@outlook.com

Introduction

Light Table is Chris Granger's open source IDE, written in ClojureScript.

http://www.lighttable.com/

Light Table's source can be found on github.com at:

https://github.com/LightTable/LightTable

Light Table Source

by the Numbers

Directory	File count
lt	2
lt/objs	46
lt/objs/clients	4
lt/objs/editor	2
lt/objs/langs	2
lt/objs/sidebar	4
lt/plugins	4
lt/util	7

First Impressions

The numbers point to a relatively small object-based system, and suggest a relatively flat structure.

These impressions are true, but also misleading.

Second Impressions

The first step in building Light Table from source is downloading a 51 meg tarball. That tarball expands into 155 meg of node-webkit and plugins.

The system is somewhat bigger than our first glance indicated.

To understand it, we'll have to start with node-webkit.

Part 1: Dance of the Matryoshkas

Node.js

According to its website, nodejs.org, node.js is:

> *a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.*

node.js is known for its single-threaded implementation, and the callback-heavy coding style it uses.

The Light Table source code generally isolates its use of callbacks. For example, the files module exports a synchronous set of file operations.

Node-Webkit

Node-Webkit supports plugins.

Light Table is a Node-Webkit plugin.

Node-Webkit plugins can contain nested plugins

LightTable has plugins.

Clojure support is provided by a Light Table plugin.

LightTable has user plugins.

User plugins can override system plugins.

Light Table plugins can inject code into client processes.

Node-Webkit starts from a manifest file: package.json

```
    "name": "LightTable",
    "main": "core/LightTable.html",
    "js-flags": "--harmony",
    "single-instance": true,
    "webkit": {
        "plugin": true
    },
    "chromium-args" : "--disable-threaded-compositing ...",
    "window": {
        "icon": "core/img/lticon.png",
        "width": 1024,
        "height": 700,
        "min_height": 400,
        "min_width": 400,
        "position": "center",
        "show":false,
        "toolbar": false,
        "frame": true
    }
}
```

The key "main" points to the starting webpage, *core/LightTable.html*.

Startup

LightTable.html loads the libraries:

- node_modules/lighttable/util/keyevents.js
- node_modules/lighttable/util/throttle.js
- node_modules/lighttable/bootstrap.js

before initiating LightTable with:

```
lt.objs.app.init();
```

keyevents.js and throttle.js are small JavaScript libraries for controlling event processing. bootstrap.js is Light Table compiled.

Take Away

Within a deployed LightTable system, LightTable code can be found at:

- LightTable_Home/core/node_modules/lighttable
- LightTable_Home/plugins
- LightTable_USERDIR/plugins

Light Table Architecture

- Object Definition
- Object
- Command
- Behavior
- Tag

Part 2: Objects, but not all the way down

LightTable's Object Model

An object is a map wrapped in an atom. Objects are mutable, and change in response to events.

Objects are defined by obj-defs (classes, more or less) that contain default values for the object, and an initialization function.

These are generally high-level objects; many are singletons.

From lt/obj/app.cljs:

The object definition of **app**:

```
(object/object* ::app
                :tags #{:app :window}
                :delays 0
                :init (fn [this]
                        (ctx/in! :app this)))
```

Object initialization

The creation of the singleton instance of **app**:

```
(def app (object/create ::app))
```

The object is created from the object-definition named *::app*.

The name of the object-definition becomes the ::type of the object.

An ::id is assigned.

{delays 0} is added to the new object.

The tags #{:app :window} are added to the new object.

Some additional system-managed slots (:args, :behaviors, :listeners, :tags) are allocated.

Object initialization (cont.)

The init function is run; it may modify the new object.

If the init function returns a vector, the crate library converts it into a DOM element node, and it is assigned to the :content slot. ::app does not return a content node.

Part 3: Activating the Model

Commands

From lt/objs/app.cljs:

```
(cmd/command {:command :window.new
              :desc "Window: Open new window"
              :exec (fn []
                      (let [w (open-window)]))})
```

A command is, basically, a named function. Command functions bound to keys take no arguments. Commands invoked from code (cmd/exec!) can be passed arguments.

Behaviors

From lt/objs/app.cljs:

```
(behavior ::focus-class
          :triggers #{:focus :show}
          :reaction (fn [this]
                      (dom/add-class (dom/$ :body) :active)
                      (dom/remove-class (dom/$ :body) :inactive)))
```

When this behavior is triggered, the html class :active is added to the DOM's body element, and the class :inactive is removed.

Tags

The **Duct Tape** of Light Table?

What's a Tag?

A tag is a ClojureScript keyword.

Every object has 1 or more tags. An object starts with the tags of its object-definition.

Each tag can be bound to many commands and many behaviors.

Tags can be added to objects dynamically. When a tag is added to an object, all the commands and behaviors bound to the tag are now attached to the object.

Similarly, tags can be dynamically removed from an object, resulting in removing the bound commands and behaviors.

Keymaps Bind Commands to Tags

from default.keymap:

```
{:+ {:app {"pmeta-r" [:refresh-connected-browser]
           "pmeta-shift-n" [:window.new]
```

For objects tagged **:app**, the command :window.new is bound to the key sequence "pmeta-shift-n". Only the singleton app object carries the :app tag; it has global scope.

These commands only apply in the document search input section of the sidebar:

```
{:+ {:sidebar.doc.search.input {"enter" [:docs.search.exec]
                                "esc" [:docs.search.hide]}}
```

Binding Behaviors to Tags

from default.behaviors:

```
{
 :+
 {:app [:lt.objs.clients.local/startup-with-local-client
        :lt.objs.app/focus-class]}}
```

Raising a Trigger

Events to triggers, see editor

Advanced Tags

Variations on a Theme

dynamic tags, specificity

Some Advanced Features

reloading classes, :exclusive, :throttle, :debounce

Summary

An object-definition sets the starting values for its instances.

Its name becomes their :type, and the return value of its init function becomes their :content.

An object is a map wrapped in an atom, stored in a registry.

Objects are not garbage collected, and must be explicitly destroyed.

WeakMap support is available for node-webkit (https://github.com/rogerwang/node-webkit/issues/298), and Light Table might be able to use it. It would require revalidating the object model.

Reserved object keys

::id ::type :args :content :behaviors #{} :tags #{} :triggers []
:listeners {} ::type name :children {} :name "myName"
:throttle msec :debounce msec

Key Functions

The following namespace definition is conventional in the LightTable code base:

```
(ns myPlugin
  (:require [lt.object :as object]
            [lt.objs.command :as cmd])
  (:require-macros [lt.macros :refer [behavior defui]]))
```

Object Creation

object/object* create an object definition
object/create
behavior
defui
cmd/command

Other Functions

object/raise object/refresh! object/destroy! object/merge! object/update! object/assoc-in! do not refresh

object/instances-by-type

Part 4: Follow the Flow of Control

Searching Documentation

Opening the Search sidebar

from default.keymap:

```
{:+ {:app {"ctrl-shift-d" [:docs.search.show]}
     :sidebar.doc.search.input {"enter" [:docs.search.exec]
                                "esc" [:docs.search.hide]}}}
```

from plugin/doc.cljs

```
(cmd/command {:command :docs.search.show
              :desc "Docs: Search language docs"
              :exec (fn [force?]
                      (when doc-search
                        (object/raise sidebar/rightbar :toggle doc-search {:f
orce? force?})
                        (object/raise doc-search :focus!))
                      )})
```

Raise the right sidebar and raise the search bar.

Part 5: Building Plugins

LightTable is Optimized for Writing LightTable

An expert is a person who has found out by his own painful experience all the mistakes that one can make in a very narrow field. - Niels Bohr

Plugin0

Plugin0 is intended to be the simplest demonstration plugin.

It displays some fixed content in a tab.

To make it work, though, we have to implement all the basic plugin mechanics.

Location

You can find the location of Light Table's user plugins directory by evaluating:

```
(lt.objs.files/lt-user-dir "plugins")
```

On Linux: ~/.config/LightTable/plugins/
It's different on Mac and Windows.

project.clj

```
(defproject plugin0 "0.1.0-SNAPSHOT"
  :description "plugin0: simplest LightTable plugin"
  :dependencies [[org.clojure/clojure "1.5.1"]])
```

No cljsbuild directives!

We will build the plugin from Light Table, which will just compile and combine the plugin ClojureScript code into "plugin0_complled.js".

No libraries will be added to the compiled JavaScript. The compiled JavaScript will be loaded into LightTable, and have access to Light Table's copy of the support libraries (e.g., the ClojureScript runtime).

Manifest

plugin.edn or plugin.json

This name is fixed.

```
{:name "plugin0"
 :version "0.0.1"
 :author "DesignTable"
 :source "https://github.com/DesignTable/plugin0"
 :desc "Simplest plugin for Light Table"
 :dependencies []
 :behaviors "plugin0.behaviors"}
```

Note the key "behaviors".

LightTable will load the referenced behavior file when it loads the plugin.

Behaviors

plugin0.behaviors -- referenced in the manifest

```
{:+ {:app [(:lt.objs.plugins/load-js ["plugin0_compiled.js"])
           (:lt.objs.plugins/load-keymap "plugin0.keymap")]
     :plugin0.panel [:lt.plugins.plugin0/destroy-on-close]}
}
```

Every plugin should start by loading its compiled javascript and its keymap.

The last line scopes the *destroy on close* behavior to the "class" *plugin0.panel*

Keymap

plugin0.keymap -- referenced in plugin0.behaviors

```
{:+ {:app {"ctrl-c 0" [:lt.plugins.plugin0/start-plugin0]}
     }
}
```

Anywhere in the app the key sequence *ctrl-c 0* will run the command *start-plugin0*

Plugin Source

from src/lt/plugins:
namespace declaration:

```
(ns lt.plugins.plugin0
  (:require [lt.object :as object]
            [lt.objs.command :as cmd]
            [lt.objs.tabs :as tabs])
  (:require-macros [lt.macros :refer [behavior defui]]))
```

Plugin Source (cont.)

```
(defui plugin0-panel [this]
  [:h1 "Plugin0: Hello!"])

; from browser
(behavior ::destroy-on-close
                  :triggers #{:close}
                  :reaction (fn [this]
                              ;(println "p0 destroy")
                              (object/destroy! this)))

(object/object* ::plugin0.panel
                :tags [:plugin0.panel]
                :name "plugin0"
                :init (fn [this]
                        (plugin0-panel this)))

(cmd/command {:command ::start-plugin0
              :desc "plugin0: Raise window"
```