LEARNING HAMMERSPOON

Unleash the power of automation on your mac

Diego Zamboni

Learning Hammerspoon

Unleash the power of automation on your Mac

Diego Zamboni

This book is for sale at http://leanpub.com/learning-hammerspoon

This version was published on 2020-08-10

Tweet This Book!

Please help Diego Zamboni by spreading the word about this book on Twitter!

The suggested hashtag for this book is #learning-hammerspoon.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#learning-hammerspoon

Also By Diego Zamboni

Learning CFEngine

Utilerías de Unix

Literate Configuration

Publishing with Emacs, Org-mode and Leanpub

Para Susi, Kari, Fabi y Nube

Contents

1. Preface to the book sample

Thank you for downloading this book sample! In it you get to key chapters of the book which will help you get started with Hammerspoon and the use of Spoons to make life on your Mac easier.

I hope you will find it useful, and encourage you to get the full book to learn a lot more about advanced uses of Hammerspoon, including how to write your own Hammerspoon configuration in Lua, and how to develop your own Spoons.

In addition to this book sample, please take a look at ther Hammerspoon-related articles in my blog at https://zzamboni.org/tags/hammerspoon/, and at my "Hammerspoon" channel in YouTube, where you will find short videos that explain in a hands-on way some of the concepts that you find in this book: https://www.youtube.com/playlist?list=PLTZ6fO4RcbeOCZQ

If you have any feedback or questions about this book, please visit the "Email the Author" page at https://leanpub.com/learning-hammerspoon/email_author/new.

Follow me on Twitter at https://twitter.com/zzamboni for more updates.

Preface to the early release

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Release notes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

August 2020

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

December 2019

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

November 2019

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

August 2019

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

April 8th, 2019

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

October 2018

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Mac automation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

What will you learn?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Conventions Used in This Book

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Getting started with Hammerspoon

What is Hammerspoon?

Hammerspoon is a Mac application that allows you to achieve an unprecedented level of control over your Mac. Hammerspoon enables interaction with the system at multiple layers–from low-level file system or network access, mouse or keyboard event capture and generation, all the way to manipulating applications or windows, processing URLs and drawing on the screen. It also allows interfacing with AppleScript, Unix commands and scripts, and other applications. Hammerspoon configuration is written in Lua, a popular embedded programming language.

Using Hammerspoon, you can replace many stand-alone Mac utilities for controlling or customizing specific aspects of your Mac (the kind that tends to overcrowd the menubar). For example, the following are doable using Hammerspoon (these are all things I do with it on my machine - you can see the configuration for these in my own Hammerspoon config file):

- Add missing or more convenient keyboard shortcuts to applications, even for complex multi-step actions. For example: automated tagging and filing in Evernote, mail/note archival in Mail, Outlook and Evernote, filing items from multiple applications to OmniFocus using consistent keyboard shortcuts, or muting/unmuting a conversation in Skype.
- Open URLs in different browsers based on regular expression patterns. When combined with Site-specific Browsers (I use Epichrome), this allows for highly flexible management of bookmarks, plugins and search configurations.

- Replace Spotlight, Lacona and other launchers with a fully configurable, extensible launcher, which allows not only to open applications, files and bookmarks, but to trigger arbitrary Lua functions.
- Manipulate windows using keyboard shortcuts to resize, move and arrange them.
- Set up actions to happen automatically when switching between WiFi networks–for example for reconfiguring proxies in some applications.
- Keyboard-triggered translation of selected text between arbitrary human languages.
- Keep a configurable and persistent clipboard history.
- Automatically pause audio playback when headphones are unplugged.

Hammerspoon is the most powerful Mac automation utility I have ever used. If you are a programmer, it can make using your Mac vastly more fun and productive.

How does Hammerspoon work?

Hammerspoon acts as a thin layer between the operating system and a Lua-based configuration language. It includes extensions for querying and controlling many aspects of the system. Some of the lower-level extensions are written in Objective-C, but all of them expose a Lua API, and it is trivial to write your own extensions or modules to extend its functionality.

From the Hammerspoon configuration you can also execute external commands, run AppleScript or JavaScript code using the OSA scripting framework, establish network connections and even run network servers; you can capture and generate keyboard events, detect network changes, USB or audio devices being plugged in or out, changes in screen or keyboard language configuration; you can draw directly on the screen to display whatever you want; and many other things. Take a quick look at the Hammerspoon API index page to get a feeling of its

extensive capabilities. And that is only the libraries that are built into Hammerspoon. There is an extensive and growing collection of Spoons, modules written in pure Lua that provide additional functionality and integration. And of course, the configuration is simply Lua code, so you can write your own code to do whatever you want.

Interested? Let's get started!

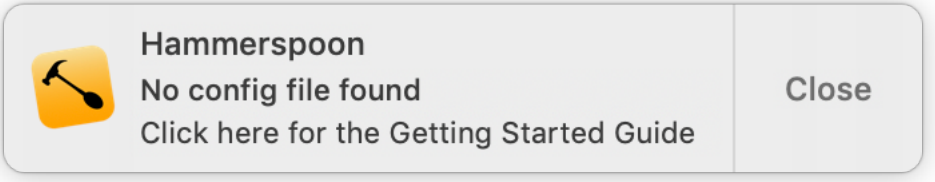
Installing Hammerspoon

Hammerspoon is a regular Mac application. To install it by hand, you just need to download it from https://github.com/Hammerspoon/hammerspoon/releases/latest, unzip the downloaded file and drag it to your /Applications folder (or anywhere else you want).

If you are automation-minded like me, you probably use Homebrew and its plugin Cask to manage your applications. In this case, you can use Cask to install Hammerspoon:

```
brew cask install hammerspoon
```

When you run Hammerspoon for the first time, you will see its icon appear in the menubar, and a notification telling you that it couldn't find a configuration file. Let's fix that!



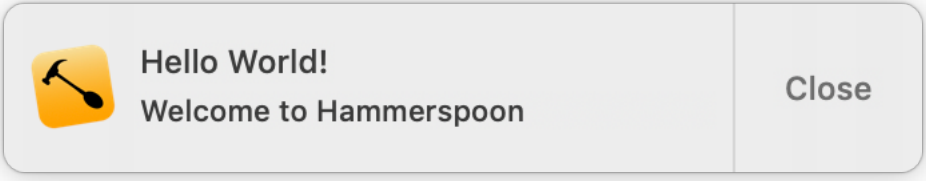
If you click on the initial notification, your web browser will open to the excellent *Getting Started with Hammerspoon* page, which I highly recommend you read for more examples.

Your first Hammerspoon configuration

Let us start with a few simple examples. As tradition mandates, we will start with a "Hello World" example. Open `$HOME/.hammerspoon/init.lua` (Hammerspoon will create the directory upon first startup, but you need to create the file) in your favorite editor, and type the following:

```
hs.hotkey.bindSpec({ { "ctrl", "cmd", "alt" }, "h" },
  function()
    hs.notify.show("Hello World!", "Welcome to Hammerspoon", "")
  end
)
```

Save the file, and from the Hammerspoon icon in the menubar, select "Reload config". Apparently nothing will happen, but if you then press `Ctrl-Alt-⌘-h` on your keyboard, you will see a notification on your screen welcoming you to the world of Hammerspoon.



Although it should be fairly self-explanatory, let us dissect this example to give you a clearer understanding of its components:

- All Hammerspoon built-in extensions start with `hs`. In this case, `hs.hotkey` is the extension that handles keyboard bindings. It allows us to easily define which functions will be called in response to different keyboard combinations. You can even differentiate between the keys being pressed, released or held down if you need to. The other extension used in this example is `hs.notify`, which allows us to interact with the macOS Notification Center to display, react and interact with notifications.

- Within `hs.hotkey`, the `hs.hotkey.bindSpec` function allows you to bind a function to a pressed key. Its first argument is a key specification which consists of a list (Lua lists and table literals are represented using curly braces) with two elements: a list of the key modifiers, and the key itself. In this example, `{ { "ctrl", "cmd", "alt" }, "h" }` represents pressing `Ctrl-Alt-⌘-h`.
- The second argument to `bindSpec` is the function to call when the key is pressed. Here we are defining an inline anonymous function using `function() ... end`.
- The callback function uses `hs.notify.show` to display the message. Take a quick look at the `hs.notify` documentation to get an idea of its extensive capabilities, including configuration of all aspects of a notification's appearance and buttons, and the functions to call upon different user actions.

Try changing the configuration to display a different message or use a different key. After every change, you need to instruct Hammerspoon to reload its configuration, which you can do through its menubar item (although we will learn how to automate it below).

The Hyper key

You will notice through this book that we use the `Ctrl-Alt-⌘` combination very frequently in our keybindings. The idea behind this is to use a modifier key combination which is never used by other applications, so that we can setup global Hammerspoon keybindings without worrying about conflicts with application-specific key bindings.

To avoid having to type `{"ctrl","alt","cmd"}` every time in the configuration file, we can define them as variable. For example, I have the following at the top of my `init.lua`:

```
hyper = { "ctrl", "alt", "cmd" }
shift_hyper = { "shift", "ctrl", "alt", "cmd" }
```

Then we can simply use `hyper` or `shift_hyper` in our key binding declarations. The example above becomes:

```
hs.hotkey.bindSpec({ hyper, "h" },
  function()
    hs.notify.show("Hello World!", "Welcome to Hammerspoon", "")
  end
)
```

I find `Ctrl-Alt-⌘` handy because the three keys are next to each other in a row right next to the spacebar in my keyboard, so I can easily hit them as a chord. You are of course free to use a different combination depending on your preferences and your keyboard layout.

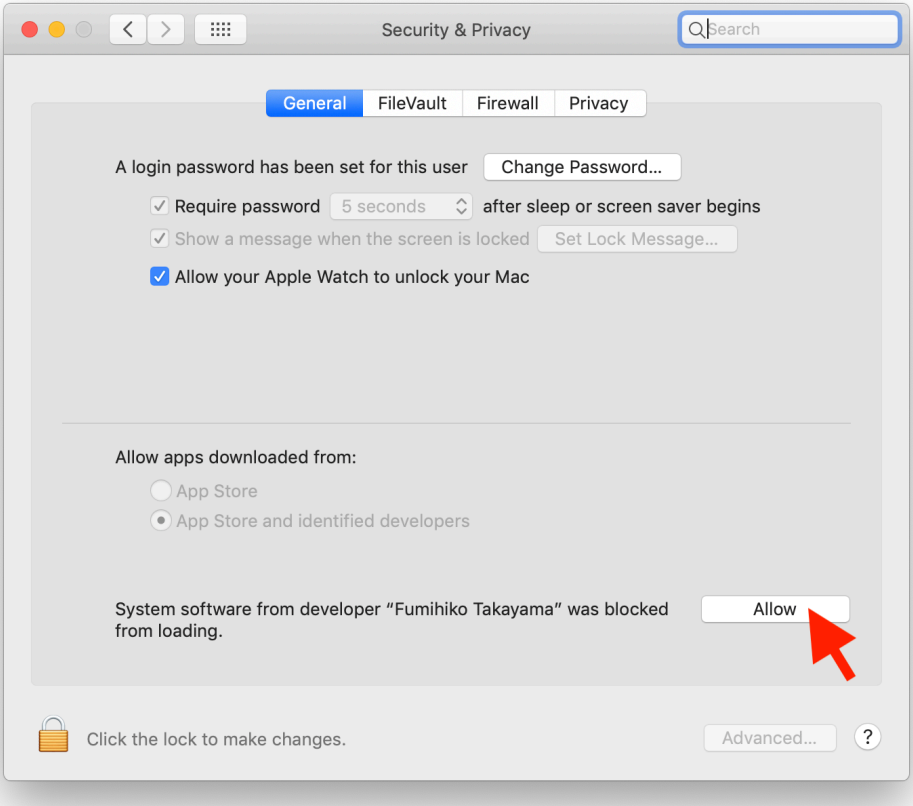
All the keybinding examples in this book will assume you have defined the `hyper` variable to represent the modifier key combination you want to use for most of your global keybindings.

Mapping a single key as Hyper using Karabiner Elements

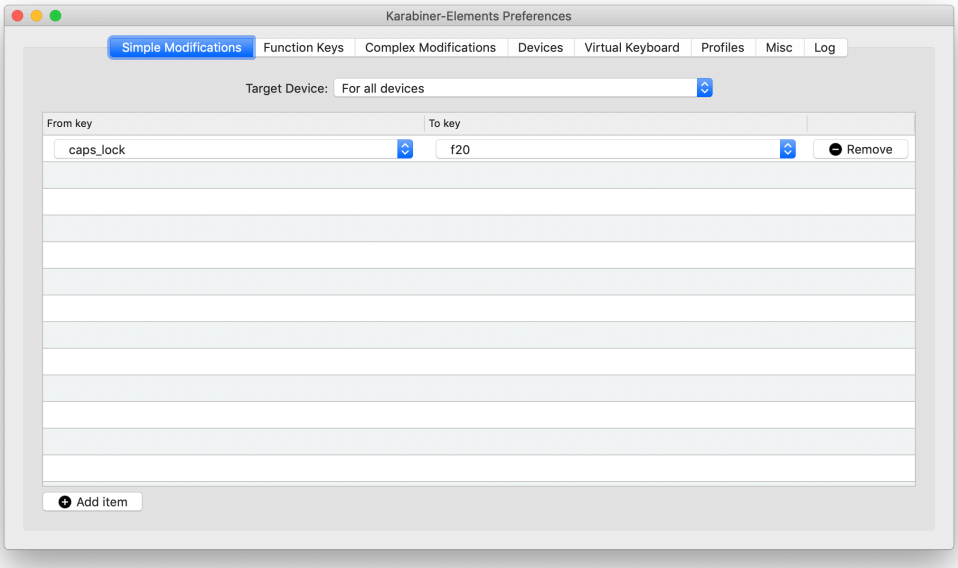
If you have a real, physical key to spare in your keyboard, you may want to map it as Hyper. For example, some people like to use the Caps Lock key as Hyper (I remap my Caps Lock key as a second Ctrl key, which I find more useful). To achieve this, you can use another free utility called Karabiner Elements, which allows you to do low-level keyboard remapping with use. You first need to install Karabiner:

```
brew cask install karabiner-elements
```

Karabiner needs to install a kernel extension to do its work, and recent versions of macOS will block it by default. You will get a dialog notifying you about this, and asking you to use the Security Preferences Pane to allow it if you want. Once you click “Allow” in this pane, Karabiner should be ready to use:



Once you run the Karabiner-Elements app, you can remap the Caps Lock key to any other key. Within the "Simple Modifications" tab you could, for example, remap Caps Lock to a nonexisting function key such as `F20`:



You need to map the `hyper` and `shift_hyper` variables in your Hammerspoon configuration according to the key you used. For example:

```
hyper = { "f20" }
shift_hyper = { "shift", "f20" }
```

Afterwards, you can use `hyper` and `shift_hyper` in your keybindings as shown before.

Keeping private information separate

It makes sense to keep your configuration files (as you should most other files) under control of a version control system like Git or Mercurial. This allows you to keep track of changes you make to your files, and it also makes it easy to share your configuration with others, for example by keeping them in Github or BitBucket.

However, it is also common to have in your configuration pieces of information that you do not want to share publicly: passwords, authentication

tokens, or simply experimental code that you are not ready to share yet. In theses cases, you can keep some configuration in separate files that are not committed to your shared files. In Lua, you can read an external file as code using the `dofile()` function. You can have a "local-only" configuration file which is read from your main `init.lua` file:

```
local localfile = hs.configdir .. "/init-local.lua"
if hs.fs.attributes(localfile) then
  dofile(localfile)
end
```

A couple of noteworthy points about this code:

- We use the `hs.configdir` variable instead of hardcoding the path. This ensures that the code will execute properly even if (for some reason) the configuration directory is stored somewhere else.
- The `dofile()` function throws an error if the file contains a syntax error which we want, but also if the file does not exist, which we do not want. For this reason we enclose the call to `dofile` in a check for existence of the file. Lua does not have a function to explicitly check for file existence, but we can use `hs.fs.attributes`, which returns `nil` if the file cannot be found.

Debugging tools and the Hammerspoon console

As you start modifying your configuration, errors will happen, as they always do when coding. To help in development and debugging, Hammerspoon offers a console window where you can see any errors and messages printed by your Lua code as it executes, and also type code to be evaluated. It is a very useful tool while developing your Hammerspoon configuration.

To invoke the console, you normally choose "Console…" from the Hammerspoon menubar item. However, this is such a common operation, that you might find it useful to also set a key combination for showing the

console. Most of Hammerspoon's internal functionality is also accessible through its API. In this case, looking at the documentation for the main `hs` module reveals that there is an `hs.toggleConsole` function. Using the knowledge you have acquired so far, you can easily configure a hotkey for opening and hiding the console:

```
hs.hotkey.bindSpec({ hyper, "y" }, hs.toggleConsole)
```

Once you reload your configuration, you should be able to use `Ctrl-Alt-⌘-y` to open and close the console. Any Lua code you type in the Console will be evaluated in the main Hammerspoon context, so you can add to your configuration directly from there. This is a good way to incrementally develop your code before committing it to the `init.lua` file.

You may have noticed by now another common operation while developing Hammerspoon code: reloading the configuration, which you normally have to do from the Hammerspoon menu. So why not set up a hotkey to do that as well? Again, the `hs` module comes to our help with the `hs.reload` method:

```
hs.hotkey.bindSpec({ hyper, "r" }, hs.reload)
```

Another useful development tool is the `hs` command, which you can run from your terminal to get a Hammerspoon console. To install it, you can use the `hs.ipc.cliInstall` function, which you can just add to your `init.lua` file to check and install the command every time Hammerspoon runs.

The `hs.ipc.cliInstall` function creates symlinks under `/usr/local/` to the `hs` command and its manual page file, located inside the Hammerspoon application bundle. Under some circumstances (particularly if you build Hammerspoon from source, or if you install different versions of it), you may end up with broken symlinks. If the `hs` command stops working and `hs.ipc.cliInstall()` doesn't fix it, look for broken symlinks left behind from old versions of Hammerspoon. Remove them and things should work again.

Now you have all the tools for developing your Hammerspoon configuration.

Using Spoons in Hammerspoon

Spoons are modules written in Lua which can be easily installed and loaded into Hammerspoon to provide ready-to-use functionality. Spoons provide a predefined API to configure and use them. They are also a good way to share your own work with other users.

Using a Spoon to locate your mouse

As a first example, we will use the MouseCircle spoon, which allows us to set up a hotkey that displays a color circle around the current location of the mouse pointer for a few seconds, to help you locate it.

To install the spoon, download its zip file from https://www.hammerspoon.org/Spoons/MouseCircle.html, unpack it, and double-click on the resulting `MouseCircle.spoon` file. Hammerspoon will install the Spoon under `~/.hammerspoon/Spoons/`.

Once a Spoon is installed, you need to use the `hs.loadSpoon()` function to load it. Type the following in the Hammerspoon console, or add it to your `init.lua` file and reload the configuration:

```
hs.loadSpoon("MouseCircle")
```

After a spoon is loaded, and depending on what it does, you may need to configure it, assign hotkeys, and start it. A spoon's API is available through the `spoon.<SpoonName>` namespace. To learn the API you need to look at the spoon documentation page. In the case of MouseCircle, a look at http://www.hammerspoon.org/Spoons/MouseCircle.html reveals that it has two methods (`bindHotkeys()` and `show()`) and one configuration variable (`color`) available under `spoon.MouseCircle`.

The first API call is `spoon.MouseCircle:bindHotkeys()`, which allows us to set up a hotkey that shows the mouse locator circle around the location of the mouse pointer. Let's say we wanted to bind the mouse circle to `Ctrl-Alt-⌘-d`. According to the MouseCircle documentation, the name for this action is `show`, so we can do the following:

```
spoon.MouseCircle:bindHotkeys({
  show = { hyper, "d" }
})
```

See The "Hyper" key for instructions on how to set up the `hyper` variable, if you have not done so yet.

Once you do this, press the hotkey and you should see a red circle appear around the mouse cursor, and fade away after 3 seconds.

All spoons which offer the possibility of binding hotkeys have to expose it through the same API:

```
spoon.SpoonName:bindHotkeys({ action1 = keySpec1,
                              action2 = keySpec2, ... })
```

Each `actionX` is a name defined by the spoon, which refers to something that can be bound to a hotkey, and each `keySpecX` is a table with two elements: a list of modifiers and the key itself, such as `{ { "ctrl", "cmd", "alt" }, "d" }` (or equivalently, `{ hyper, "d"}`)

The second API call in the MouseCircle spoon is `show()`, which triggers the functionality of showing the locator circle directly. Let's try it! Type the following in the console:

```
spoon.MouseCircle:show()
```

Most spoons are structured like this: you can set up hotkeys to trigger the main functionality, but you can also trigger it via method calls.

Normally you won't use these methods, but their availability makes it possible for you to use spoon functionality from our own configuration, or from other spoons, to create further automation.

`spoon.MouseCircle.color` is a public configuration variable exposed by the spoon, which specifies the color that will be used to draw the circle. Colors are defined according to the documentation for the `hs.drawing.color` module. Several color collections are supported, including the OS X system collections and a few defined by Hammerspoon itself. Color definitions are stored in Lua tables indexed by their name. For example, you can view the `hs.drawing.color.hammerspoon` table, including the color definitions, by using the convenient `hs.inspect` method on the console:

```
> hs.inspect(hs.drawing.color.hammerspoon)
{
  black = {
    alpha = 1,
    blue = 0.0,
    green = 0.0,
    red = 0.0
  },
  green = {
    alpha = 1,
    blue = 0.0,
    green = 1.0,
    red = 0.0
  },
  osx_red = {
    alpha = 1,
    blue = 0.302,
    green = 0.329,
    red = 0.996
  },
  osx_green = {
...
```

Lua does not include a function to easily get the keys of a table so you have to use the `pairs()` function to loop over the key/value pairs of the table. The `hs.inspect` function is convenient, but to get just the list of tables and the color names, without the color definitions themselves, you can use the following code (if you type this in the console you have to type it all in a single line – and beware, the output is a long list):

```
for listname,colors in pairs(hs.drawing.color.lists()) do
  print(listname)
  for color,def in pairs(colors) do
    print("  " .. color)
  end
end
```

If we wanted to make the circle green, we can assign the configuration value like this:

```
spoon.MouseCircle.color = hs.drawing.color.hammerspoon.green
```

The next time you invoke the `show()` method, either directly or through the hotkey, you will see the circle in the new color.

You may have noticed that we accessed the configuration variable with a dot (`spoon.MouseCircle.color`), and we also used it for some function calls (e.g. `hs.notify.show`, whereas for `show()` we used a colon (`spoon.MouseCircle:show()`). The latter is Lua's object-method-call notation, and its effect is to implicitly pass the object as an implicit first argument called `self`. This is simply a syntactic shortcut, i.e. the following two are equivalent:

```
spoon.MouseCircle:show()
spoon.MouseCircle.show(spoon.MouseCircle)
```

Normally you would use colon notation, but the alternative can be useful when constructing function pointers. For example, if you wanted to bind a second key to show the mouse circle, you might initially try the following:

```
hs.hotkey.bindSpec({ hyper, "p" },
                   spoon.MouseCircle:show)
```

But this results in an error. The correct way is to wrap the call in an anonymous function:

```
hs.hotkey.bindSpec({ hyper, "p" },
                   function() spoon.MouseCircle:show() end)
```

Alternatively, you can use the `hs.fnutils.partial` function to construct a function pointer that includes the correct first argument:

```
hs.hotkey.bindSpec({ hyper, "p" },
                   hs.fnutils.partial(spoon.MouseCircle.show,
                                      spoon.MouseCircle))
```

Lua supports using functions as first-class values, and the `hs.fnutils` extension includes a number of functions that make it easy to use them.

By now you know enough to use spoons with Hammerspoon's native capabilities: look for the ones you want, download and install them by hand, and configure them in your `init.lua` using their configuration

variables and API. In the next section we will explore how to install and configure spoons in a more automated way.

Automated Spoon installation and configuration

Once you develop a complex Hammerspoon configuration using spoons, you may start wondering if there is an easy way to manage them. There are no built-in mechanisms for automatically installing spoons, but you can use a spoon called SpoonInstall that implements this functionality. You can download it from http://www.hammerspoon.org/Spoons/SpoonInstall.html. Once installed, you can use it to declaratively install, configure and run spoons. For example, with SpoonInstall you can use the MouseCircle spoon as follows:

```
hs.loadSpoon("SpoonInstall")
spoon.SpoonInstall:andUse("MouseCircle", {
                  config = {
                     color = hs.drawing.color.osx_red,
                  },
                  hotkeys = {
                     show = { hyper, "d" }
                  }})
```

If the MouseCircle spoon is not yet installed, `spoon.SpoonInstall:andUse()` will automatically download and install it, and set its configuration variables and hotkeys according to the declaration.

If there is nothing to configure in the spoon, `spoon.SpoonInstall:andUse("SomeSpoon")` does exactly the same as `hs.loadSpoon("SomeSpoon")`. But if you want to set configuration variables, hotkey bindings or other parameters, the following keys are recognized in the map provided as a second parameter:

- `config` is a Lua table containing keys corresponding to configuration variables in the spoon. In the example above, `config = {`

`color = hs.drawing.color.osx_red }` has the same effect as setting `spoon.MouseCircle.color = hs.drawing.color.osx_red`

- `hotkeys` is a Lua table with the same structure as the mapping parameter passed to the `bindHotkeys` method of the spoon. In our example above, `hotkeys = { show = { hyper, "d" } }` automatically triggers a call to `spoon.MouseCircle:bindHotkeys({ show = { hyper, "d" } })`.
- `loglevel` sets the log level of the `logger` attribute within the spoon, if it exists. The valid values for this attribute are 'nothing', 'error', 'warning', 'info', 'debug', or 'verbose'.
- `start` is a boolean value which indicates whether to call the Spoon's `start()` method (if it has one) after configuring everything else.
- `fn` specifies a function which will be called with the freshly-loaded Spoon object as its first argument. This can be used to execute other startup or configuration actions that are not covered by the other attributes. For example, if you use the Seal spoon (a configurable launcher), you need to call its `loadPlugins()` method to specify which Seal plugins to use. You can achieve this with something like this:

```
spoon.SpoonInstall:andUse("Seal",
  { hotkeys = { show = { {"cmd"}, "space" } },
    fn = function(s)
             s:loadPlugins({"apps", "calc", "safari_bookmarks"})
         end,
         start = true,
  })
```

- `repo` indicates the repository from where the Spoon should be installed if needed. Defaults to `"default"`, which indicates the official Spoon repository at http://www.hammerspoon.org/Spoons/. I keep a repository of unofficial Spoons at http://zzamboni.github.io/zzSpoons/, and others may be available by the time you read this.
- `disable` can be set to `true` to disable the Spoon (easier than commenting it out when you want to temporarily disable a spoon) in your configuration.

You can assign functions and modules to variables to improve readability of your code. For example, in my `init.lua` file I make the following assignment:

```
Install=spoon.SpoonInstall
```

Which allows me to write `Install:andUse("MouseCircle",...)`, which is shorter and easier to read.

Managing repositories and spoons using SpoonInstall

Apart from the `andUse()` "all-in-one" method, SpoonInstall has methods for specific repository- and spoon-maintenance operations. As of this writing, there are two Spoon repositories: the official one at http://www.hammerspoon.org/Spoons/, and my own at http://zzamboni.github.io/zzSpoons/, where I host some unofficial and in-progress Spoons.

The configuration variable used to specify repositories is `SpoonInstall.repos`. Its default value is the following, which configures the official repository identified as "default":

```
{
  default = {
      url = "https://github.com/Hammerspoon/Spoons",
      desc = "Main Hammerspoon Spoon repository",
  }
}
```

To configure a new repository, you can define an extra entry in this variable. The following code creates an entry named "zzspoons" for my Spoon repository:

```
spoon.SpoonInstall.repos.zzspoons = {
   url = "https://github.com/zzamboni/zzSpoons",
   desc = "zzamboni's spoon repository",
}
```

After this, both “zzspoons” and “default” can be used as values to the `repo` attribute in the `andUse()` method, and in any of the other methods that take a repository identifier as a parameter. You can find the full API documentation at http://www.hammerspoon.org/Spoons/SpoonInstall.html.

Just enough Lua to be productive with Hammerspoon

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Flow control

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Dot-vs-colon method access in Lua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Tables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Tables as namespaces

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

String manipulation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Learning more Lua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

DRAFT Exploring the Hammerspoon API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Events and Hotkeys

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

`hs.hotkey` and `hs.hotkey.modal`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

`hs.keycodes` and `hs.keycodes.map`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

`hs.eventtap` and `hs.eventtap.event`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

`hs.mouse`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Window, Menus and Screen Manipulation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

`hs.screen`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

`hs.window` and other window-manipulation libraries

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Other modules

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

On-screen Drawing, Images and Alerts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Application and Process Manipulation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Sound and Music

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Networking and Web

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

System and Device Manipulation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Data Processing and Utilities

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Hammerspoon itself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

DRAFT Hammerspoon cookbook, tips and tricks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Tip: be mindful of garbage collection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Show Homebrew package info

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Tip: using asynchronous methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Transform URLs before opening them

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Other resources and configuration examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Spacehammer: a Spacemacs-like modal config

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Writing your own extensions and Spoons

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Writing a new Spoon

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

The Spoon API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Designing a new spoon: Leanpub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Creating the skeleton for a new spoon

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Spoon metadata and configuration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Spoon methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Testing your Spoon

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Packaging and sharing the new Spoon

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

What's next?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Writing a Hammerspoon extension in Lua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.

Using and extending Seal

Using Seal

Writing your own Seal plugins

Seal plugin structure and API

The `myactions` Seal plugin

Adding commands to your Seal plugin

Colophon

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/learning-hammerspoon.