Reactive Programming with JavaScript

Reactive programming is carried out using the building blocks of functional programming. JavaScript libraries such as ReactJS are used for frontend web development that is both competent and powerful.

This book is among the first to address how everyday programmers can take advantage of (functional) reactive programming without having an extremely heavy mathematical background. Along with the basics a frontend developer can easily connect with, it also covers the basics of functional programming, then goes on to explain nonfunctional reactive programming using a live example, and finally finishes with building one small and one larger frontend project.

Who this book is written for

If you are proficient with JavaScript and want to know about functional, reactive, or functional reactive programming and the Facebook approach to it, then this book is for you. This book is also for web frontend developers who want to see how programming is fun again! A basic knowledge of JavaScript is expected.

What you will learn from this book

- Learn functional reactive programming with JavaScript for non-mathematicians
- Experience Facebook’s primary frontend framework, ReactJS
- An excellent stepping stone to learning ReactJS Native
- Using the tools Facebook uses to build a better site in less time
- Create and implement a website using Node.js
- Dive into the development of web apps using ReactJS
- Implementation of FRP ReactJS with live examples

Tools to make functional reactive programming easier such as Bacon.js, a library like jQuery, are also covered.

Finally, it finishes with building one small and one larger frontend project.


Free Sample
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Introduction and Installation'
- A synopsis of the book’s content
- More information on *Reactive Programming with JavaScript*
About the Author

Jonathan Hayward is a polymath with advanced degrees bridging mathematics, computers (UIUC), theology, and philosophy. He obtained his theology and philosophy degrees from Cambridge University. He has worked in many areas of web development, with a site (http://cjsh.name) for "after hours" titles, and he is also interested in the human side of computing, including usability/UI/UX. His most popular work is a piece of poetry at https://cjshayward.com/doxology. The faster route to get there is by typing cjsh.name/doxology, and it gets there. Jonathan has studied many languages, including French, Spanish, Latin, and Greek. He is currently learning Russian. He has worked on various other books as well (refer to http://www.amazon.com/s/ref=nb_sb_noss_2?url=search-alias%3Daps&field-keywords=%22CJS+Hayward%22 to find out more).
Charles Cézanne famously said about the impressionist painter Claude Monet, "Monet is only an eye, but what an eye!" Today, we can similarly say, "ReactJS [or if you prefer, "ReactJS is only a view, but what a view!""]

ReactJS has neither the intention nor the ambition to be a complete, general-purpose web framework. It doesn't even include tooling for Ajax calls! Rather, the intent is that you will use technologies that make sense for different concerns in your application, and use ReactJS's power tools for views and user interface development.

Functional reactive programming has been an extremely high-hanging fruit, with a prohibitive barrier to entry in terms of sheer mathematical expectations assumed in order to work with it. No longer with ReactJS! A veteran C++ programmer with no particularly deep math background—I said this to pick a profile of programmers who keep on saying on Stack Overflow that they don't get functional reactive programming—is a veteran programmer who stands a fair chance of getting real work done using ReactJS.

This book is about ReactJS, a simple and small technology that nonetheless lets huge teams work together on different components of a web page without stepping on each others' feet, but without a hint of bureaucratic measures. And add some liberal help of pixie dust.

**What this book covers**

*Chapter 1, Introduction and Installation,* provides a 10,000-foot overview of different programming paradigms, each of which has its strengths, and an introduction to the trio of functional programming, reactive programming, and functional reactive programming.
Chapter 2, Core JavaScript, covers some of JavaScript's better neighborhoods and omits the minefields, with a debt to Douglas Crockford, if not a complete agreement. In terms of the parts of JavaScript that you use, you should be doing most of your work within this core.

Chapter 3, Reactive Programming – The Basic Theory, is a basic exploration of reactive theory, or reactive programming, specifically in relation to Facebook's ReactJS user interface framework.

Chapter 4, Demonstrating Nonfunctional Reactive Programming – A Live Example, proves that not all development is from scratch. Most professional work is not greenfield. This will offer a live example of retrofitting a simple video game, the most recent implementation using jQuery, to take advantage of ReactJS (if you are using ReactJS, you will probably be doing other conversions from jQuery to ReactJS).

Chapter 5, Learning Functional Programming – The Basics, helps you if you want to understand functional programming but have no idea where to start. Here's one place to start! Map, reduce, and filter are introduced as an inexhaustible bag of tricks.

Chapter 6, Functional Reactive Programming – The Basics, covers what has been said about functional programming and reactive programming. It will be put together with some sage advice, and the last bit of foundation will be laid for the remaining hands-on work in this book.

Chapter 7, Not Reinventing the Wheel – Tools for Functional Reactive Programming, contains a lot to cover in one book, let alone one chapter. But there is meant to be an interesting sampling of a space where a lot of interesting options are made available, including writing ReactJS code from a language other than JavaScript.

Chapter 8, Demonstrating Functional Reactive Programming in JavaScript – A Live Example, Part I, is where we see an application housing a whimsical ReactJS component written in ReactJS from scratch, and showcasing the sweet JSX syntactic sugar that is not required but is still made available for ReactJS development.

Chapter 9, Demonstrating Functional Reactive Programming in JavaScript with a Live Example Part II – A To-do List, leads us to our first real component, designed to be used rather than just amuse. We implement a to-do list, and it has several markers other than just "done" to indicate what state, priority, and other things a task has.

Chapter 10, Demonstrating Functional Reactive Programming in JavaScript: A Live Example Part III – A Calendar, is where we build a calendar. It is intended to gracefully support not only one-time events but also many kinds of repeating events with all kinds of rules that people offer.
Chapter 11, Demonstrating Functional Reactive Programming in JavaScript with a Live Example Part IV – Adding a Scratchpad and Putting It All Together, provides a rich-text scratchpad with CKeditor. This shows how we might interoperate with other user interface tools. Then we wind up by bringing the four components together into one combined page and adding persistence functionality so that our user interface doesn't forget what it is told.

Chapter 12, How It All Fits Together, reviews what we covered in this book, and we look at the next steps in a world to explore.

Appendix, A Node.js Kick start, looks at some of the good, the bad, and the ugly of a "Wild, Wild West" technology that everybody seems to want in on.
Introduction and Installation

Welcome to the wonderful world of reactive (functional) programming in JavaScript! In this book, we will cover the good parts of JavaScript, although we will not follow it religiously. We will cover functional programming, reactive programming, and the ReactJS library and integrate all of these into functional reactive programming with JavaScript. If you are going to study reactive programming, it is our suggestion that you seriously consider functional reactive programming, including learning functional programming as much you can. In this context, the whole of functional reactive programming is more than the sum of its parts. We will be applying reactive programming to JavaScript user interface development. User interfaces are one domain in which functional reactive programming (FRP), really shines.

The topics that will be covered in this chapter are as follows:

- A 10,000-foot overview of the subject-matter, including:
  - A discussion of an easier approach to user interface programming
  - A brief discussion of programming paradigms, such as functional and reactive programming
- An overview of the chapters in this book
- A look at how to install some of the tools used over the course of this book

A 10,000-foot overview

There are a lot of things that can be stated, but (functional) reactive programming may be easier than you think. Today, much that has been written about functional reactive programming is intimidating, much like the instructions for closures a few years ago.
An easier way to handle user interface programming

Years ago, when I first began to study JavaScript, I picked one site and literally took everything from it that I needed to understand to perform object-oriented information hiding, that is, how to create a JavaScript object that had private fields. I read perhaps two or three chapters that were dense with theoretical computer science and 10-15 percent of the introduction before giving up. Then I saw how easy it was to use a closure to create an object with private fields on a simple *monkey see, monkey do* basis:

```javascript
var counter = (function() {
    var value = 0;
    return {
        get_value: function() {
            return value;
        },
        increment_value: function() {
            value += 1;
        }
    }
})();
```

Right now, functional reactive programming is in the state in which JavaScript closures were some years back. The amount of theory you have to read before you can start reactive programming is astounding, and large portions of the literature are of a PhD reading level. That's bad news. But the good news is that you don't have to wade through so much reading.

The purpose of this book is to provide something comparable to the *monkey see, monkey do* way of conveying how to use a closure to make a JavaScript object with private fields. Theory, as such, is not bad, nor is it a problem to introduce theory for a discussion, but making a full-fledged dissertation's theoretical backing as the price to do something simple is a problem.

It is our hope that this book will let you understand why building, for instance, a game UI in JavaScript is easier with functional reactive programming than with jQuery.
Programming paradigms

There are multiple programming paradigms around, and not all are mutually exclusive. Many programming languages are multiparadigm languages, supporting the use of more than one paradigm, including not only JavaScript, but also the likes of OCaml, PHP, Python, and Perl.

Note that you can at least sometimes use a paradigm with a language that is not explicitly designed to support it. Object-oriented programming was originally formulated not for languages such as Java or Ruby that are specifically intended to support object-oriented programming, but as matter of an engineering discipline originally used in languages that predate object-oriented programming.

Among the programming paradigms, we now have the following:

• **Aspect-oriented programming**: Some have suggested that the professional development of a programmer moves from procedural programming to object-oriented programming, then to aspect-oriented programming, and finally to functional programming. A canonical example of an aspect-oriented concern, for an aspect that is spread through the program in naïve usage, is logging. Aspect-oriented programming deals with cross-cutting aspects of programming, such as security, the diagnostic exposure of a state, and logging.

• **Declarative programming**: One of the key concepts of functional reactive programming is that it is declarative rather than imperative. In other words, \( c = a + b \) does not mean take the present value of \( a \), add the present value of \( b \), and store their sum in \( c \). Instead, we declare a lasting relationship that works a bit like \( C_1 = A_1 + B_1 \) in a spreadsheet. If \( A_1 \) or \( B_1 \) changes, \( C_1 \) is immediately affected by the change. What is stored in \( C_1 \) is not the value of \( A_1 \) plus the value of \( B_1 \) at the time of assignment, but something more lasting from which individual values may be obtained in a *print on demand* fashion.

• **Defensive programming**: Analogous to defensive driving, defensive coding means writing code that behaves correctly when it is given something defective. Functional reactive programming is, among other things, an approach to either functioning correctly or degrading gracefully in the face of network issues and nonideal, real-world conditions.
• **Functional programming**: Here, the term *function* has its mathematical rather than programming meaning. In imperative programming, functions can (and most often, they do) manipulate states. Hence, an `init()` function might initialize all of the data that a program initially needs to run initially. A function is something that takes zero or more inputs and returns a result. For example, \( f(x) = 3x+1 \), \( g(x) = \sin(x) \), and \( h(x, y) = x'(y) \) (the derivative of \( x \) at \( y \)) are all mathematical functions; none of them command any manipulation of stateful data. A pure function is a function under a mathematical definition that excludes telling how to deal with states. Functional programming also allows and often includes, with the last derivative-based example, higher order functions, or functions that act on functions (in calculus, a derivative or an integral represents a higher order function, and iterative integration includes a higher order function that takes another higher order function as the input). Problems whose solutions center on abstract functions that operate on abstract functions tend to be more appealing to computer science types than something really used in the business world. The higher order functions explored here will be relatively concrete. You need not use higher order functions all the time, and once you've grasped the core concepts, they are not hard to use.

• **Imperative programming**: Imperative programming is a common way of programming, and for the majority of programmers who are first taught imperative programming, it may seem the most natural way to work. Functional reactive programming’s marketing proposal includes a live alternative to this basic approach. An alternative to the natural-seeming tendency towards imperative programming is found in functional reactive programming’s declarative programming, pure functions (including higher order functions) in functional programming, and the time series of reactive programming.

• **Information hiding**: Steve McConnel’s *Code Complete* describes several methodologies, and tells us which are optimal for different settings (the sweet spot for procedural programming is on smaller projects than on object-oriented programming, for instance). For information hiding alone, his recommendation was *use this as much as possible*. In generic information hiding developments, a large project is approached by walling off secrets within the larger area, and larger secrets are divided by walling off subsecrets. A large portion of procedural programming, object-oriented programming, and functional programming alike is intended to facilitate information hiding. Information hiding is the software engineering concern behind the Law of Demeter, for example, you may have up to one dot in a method call `(foo.bar())`, but not two `(foo.baz.bar())`. 

_______

[4]
• **Object-oriented programming**: Instead of having a monolithic architecture, a program is segmented into objects. These objects have their own methods and fields and may in turn be segmented into further objects. This offers an acceptable level of information hiding for larger projects than procedural programming, even if object-oriented programming more or less starts with procedural programming and builds on top of it.

• **Patterns**: Patterns are not a recipe for good software, but at a higher level of human abstraction, they provide a way of talking about the best recurring solutions so as to avoid reinventing from scratch what has already been solved. Also, specific patterns are taken into the limelight, including MVC and now the **Observer** pattern, which is often not mentioned in relation to reactive programming despite being a founding ingredient.

• **Procedural programming**: Procedural programming is one of the oldest of the methodologies mentioned, and it was meant to provide some order to the spaghetti code fostered by the even older **goto-based** flow control. Perhaps we can criticize procedural programming for not doing enough once object-oriented programming, aspect-oriented programming, and object-oriented design patterns are available. It is the right thing to move on from procedural programming when you have tools to push further than procedural programming from a rat's nest of gotos, the pointer as the goto of data structures, and so on.

• **Reactive programming**: Suppose functional programming is, in large measure, programming where functions have first-class status and it is possible to make higher order functions (functions that act on other functions as input). Then reactive programming is, in large measure, programming where time series (functions that have different values over time) have first-class status. For music, games, user interfaces, and some other use cases, calculating the right value for the present moment is an area where reactive programming shines.

• **Functional reactive programming**: Functional reactive programming is reactive programming built on functional building blocks, and in which both functions and time series are first-class entities. There are some useful, and surprisingly simple, functions that act on one time series to provide another time series from it (either of these series can be acted on by other functions on time series). One of the major selling points of functional reactive programming is that it provides a more graceful and much more maintainable approach than following your nose straight into the **callback hell**.
Installing the tools required
Many readers will be comfortable enough with simply installing Chrome and Node.js, if they were not already installed some time ago. For those who would prefer step-by-step directions, here are the details of installing the appropriate software.

Google Chrome can be installed from http://google.com/chrome. Note that for some Linux distributions, Chrome may or may not be available from your package manager. Google Chrome is an obvious choice to think of for something to include in a distribution's packages, but licensing concerns may list Chrome as non-free due to some of its parts, meaning that as far as the distribution maintainers are concerned, you may be welcome to use this, but we're not comfortable including it in a free-only package repository.

Node.js is available from http://nodejs.org/download. If you are running Linux, it is probably better to obtain it through your package manager. Note that Node.js comes with its own package manager, npm, which can be used to download packages that can be used under Node.js.


The following instructions are given for Windows 8.1 (I prefers to develop in Mac or Linux, but is writing for Windows 8.1 as a common lingua franca).

Installing Google Chrome
We will be using Google Chrome as the primary reference browser:

1. To download it, go to http://google.com/chrome, and click on the Download now button to the left, towards the bottom, as shown here:
2. Next, click on the **Accept and Install** button down and to the right, as shown in the following screenshot:
3. After that, click on the **Run** button when asked whether you want to run or save the installer, as shown here:

![Screenshot of ChromeSetup.exe installation prompt](image1.png)

4. Next, authorize Chrome’s installer to make changes to the system, as shown in the following screenshot:

![Screenshot of User Account Control](image2.png)
5. Then click on the **Next** button to install Chrome, as shown in the following screenshot:

![Chrome installation screenshot](image)

6. Wait a minute for it to install and then if you are willing, set Chrome as your default browser:

![Default browser selection](image)

And that's it!
Installing Node.js
Installing Node.js is straightforward and it makes it easy to start serving HTTP using JavaScript as the only language.

1. Go to http://nodejs.org/download:
2. Click on the **Windows Installer** and wait for the installer to download. Then click on the bottom-left part of the window, as shown in this screenshot:
3. When the installer starts, click on **Next**, as follows:

4. Then click on the checkbox to accept the terms of the agreement, as shown here:
5. After that, click on the **Next** button to continue, as shown in the following screenshot:

![Next button screenshot](image)

6. When asked where to install the software, click on the **Next** button, as shown in this screenshot:

![Destination Folder screenshot](image)
7. Then click on **Next** to move forward, as shown in the next screenshot. Customize the features if you want to:

![Custom Setup](image)

8. After that, click on the **Install** button to go ahead with the installation, as shown here:

![Ready to install Node.js](image)
9. Finally, click on the **Finish** button to finish installing Node.js, as shown in this screenshot:

![Node.js Setup](image)

10. Authorize the installer to make changes to your computer, shown as follows:

![User Account Control](image)
Installing the Starter Kit for ReactJS

To install the Starter Kit perform the following steps:

1. Go to http://facebook.github.io/react/downloads.html and you will see a screen similar to the following screenshot:
2. Click on **Download Starter Kit 0.12.0** to start the download (this will be visible in the bottom-left corner), as shown in the preceding screenshot.

3. You will see a ZIP file downloaded at the bottom:
4. From here, you should be able to explore the contents of the ZIP file:

Summary
In this chapter, we had a brief overview of programming paradigms to suggest where functional reactive programming might fit, and installed the basic tools.

We will be moving on to discussing JavaScript in the next chapter. The basics of Node.js are discussed in Appendix.
Where to buy this book


Click here for ordering and shipping details.