Extracted from:

Web Development with ReasonML

Type-Safe, Functional Programming for JavaScript Developers

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Web Development with ReasonML

Type-Safe, Functional Programming for JavaScript Developers

J. David Eisenberg

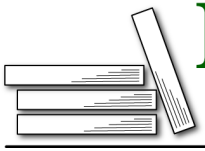
edited by Andrea Stewart

Web Development with ReasonML

Type-Safe, Functional Programming for JavaScript Developers

J. David Eisenberg

The Pragmatic Bookshelf
Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Andrea Stewart
Copy Editor: Sean Dennis
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-633-4
Book version: P1.0—April 2019

CHAPTER 7

Structuring Data with Records and Modules

While the tuple we built in Chapter 5, Using Collections, on page ? was useful and appropriate for the task, tuples aren't the answer when we have to work with more complex data structures. Instead, ReasonML has *records*, which let you create immutable data structures with field names. This makes your code more organized and readable.

Modules also help in keeping code organized. You'll see that you can create modules to hold data types, records, and functions that operate on them. Modules are also first-class citizens in the ReasonML world. You'll see this in action as we create custom modules that serve as arguments to other modules.

Specifying Records

Tuples were adequate for defining a data type for an order of shirts expressed as the quantity of shirts and the shirt size, as we saw on page ?. But there's more to shirts than just their size. You need to know whether each one is long-sleeved or short-sleeved, the color, pattern, type of cuff, and type of collar. This is definitely not a job for a tuple. There are seven pieces of data, and I'll bet if you walk away from this book for five minutes, you won't remember what order they are in.

First, let's define some data types for size, sleeve length, color, pattern, cuff, and collar:

records/shirts/src/Shirts.re

```
type size =
  | XSmall(int)
  | Small
  | Medium
  | Large
  | XLarge(int);
```

```
type sleeve =
  | Short
  | Long
  | XLong; /* for tall people */

type color =
  | White
  | Blue
  | Red
  | Green
  | Brown;

type pattern =
  | Solid
  | Pinstripe
  | Check;

type cuff =
  | Button
  | French
  | NoCuff;

type collar =
  | Button
  | Straight
  | Spread;
```

Even though `Button` appears in both `cuff` and `collar`, there's no conflict. If you have code like this:

```
let ambiguous = Button;
let explicit: cuff = Button;
```

In the first line, ReasonML's type inference will choose the last `Button` you specified (from `collar`). You can always explicitly tell ReasonML which `Button` you want by annotating your variables.

Now, we define a record type that gives all the information needed to specify a shirt. It's okay to have a field name the same as its data type. (And yes, short-sleeve shirts with French cuffs really exist.)

records/shirts/src/Shirts.re

```
type order = {
  quantity: int,
  size: size,
  sleeve: sleeve,
  color: color,
  pattern: pattern,
  cuff: cuff,
  collar: collar
};
```

Accessing and Updating Records

Here's a definition of a record of the `order` type and an example of how you access the individual fields using dot notation. You don't have to specify the fields in the same order that you used when you created the data type:

records/shirts/src/Shirts.re

```
let myOrder = {
  quantity: 1,
  size: XLarge(1),
  sleeve: Long,
  color: Blue,
  pattern: Solid,
  cuff: Button,
  collar: Button
};

Js.log2("Size:", myOrder.size); /* Size: [1, tag: 1] */
```

This *looks* a lot like a JavaScript object, but it isn't one. Let me say that again: ReasonML records are *not* JavaScript objects. We'll discuss that when we talk about Interoperating with Objects, on page ?. One of the biggest differences is that records are immutable. You can't change the value of a field in a record. Instead, you have to create a brand-new record. Looking at all those fields, you might be terribly disheartened, but don't worry. ReasonML has the *spread* operator. Here's the code to create a new order the same size as the first one, but with a different color and different style of cuff:

records/shirts/src/Shirts.re

```
let otherOrder = {
  ...myOrder,
  color: White,
  cuff: French
};

Js.log2("Cuff:", otherOrder.cuff); /* Cuff: 1 */
```

If You Really Need Mutability

Okay, maybe records aren't *that* immutable. If you absolutely, positively must have a modifiable field in a record, precede its name with the keyword mutable. As you adopt a more functional programming style, you'll find that you won't need mutability as much as you thought, so try to keep mutable to a minimum.

records/mutable-record/src/Demo.re

```
type person = {
  name: string,
  mutable age: int
};

let happyBirthday = (someone:person) : unit => {
  someone.age = someone.age + 1;
  ()
};

let friend = {
  name: "Juanita Fulano",
  age: 34
};

happyBirthday(friend);
Js.log(friend.age); /* 35 */
```