Extracted from:

Dart for Hipsters

This PDF file contains pages extracted from *Dart for Hipsters*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2012 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Dart for Hipsters

Fast, Flexible, Structured Code for the Modern Web

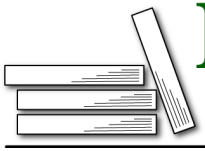
Chris Strom

Edited by Michael Swaine

Dart for Hipsters

Chris Strom

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Michael Swaine (editor)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Copyright © 2012 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-937785-03-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P4.0—January, 2014

CHAPTER 1

Project: Your First Dart Application

Most programming books start with a "Hello World!" sample. I say, screw that—we're all hipsters here. Let's start coding!

Since Dart is written, above all else, to be familiar, we should not be too far out of our depths diving right in. Let's jump straight to something more fun: an Ajax-powered website. Any true hipster has an extensive collection of comic books (am I right? I'm not the only one, am I?), so let's consider a simple Dart application that manipulates the contents of that collection via a REST-like interface.

At some point, this may prove too much of a whirlwind. Have no fear, we will go into details in subsequent chapters.

1.1 The Back End

Sample code for this chapter can be found in the "your_first_dart_app" branch of https://github.com/eee-c/dart-comics. The back end is written in Dart and requires a few Dart packages to be installed with the Dart `pub` packagers. Instructions are contained in the project's README.

Being REST-like, the application should support the following:

- `GET /comics` (return a list of comic books)
- `GET /comics/42` (return a single comic book)
- `PUT /comics/42` (update a comic book entry)
- `POST /comics` (create a new comic book in the collection)
- `DELETE /comics/42` (delete a comic book)

We will not worry too much about the details of the back end beyond that.

• Click HERE to purchase this book now. discuss

1.2 HTML for Dart

Our entire application will follow the grand tradition of recent client-side MVC frameworks. As such, we require only a single web page.

your_first_dart_app/web/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Dart Comics</title>
  <link rel="stylesheet" href="/stylesheets/style.css">

  <!-- The main application script -->
  <script src="/scripts/comics.dart" type="application/dart"></script>

  <!-- Force Dartium to start the script engine -->
  <script>
    navigator.webkitStartDart();
  </script>

</head>

<body>
  <h1>Dart Comics</h1>
  <p>Welcome to Dart Comics</p>
  <ul id="comics-list"></ul>
  <p id="add-comic">
    Add a sweet comic to the collection.
  </p>
</body>
</html>
```

Most of that web page should be familiar; it will include simple HTML, links for CSS, and scripts.

HTML Head

The only oddity to note is the first <script> tag, in which *JavaScript* starts the Dart scripting engine.

```
<!-- Force Dartium to start the script engine -->
<script>
  navigator.webkitStartDart();
</script>
```

Next we load the contents of our actual code. The only change here is a different type attribute in the <script> tag, indicating that this is Dart code.

```
<!-- The main application script -->
<script src="/scripts/comics.dart" type="application/dart"></script>
```

Important

At the time of this writing, it is necessary to kick-start the Dart VM with navigator.webkitStartDart() on Dartium, the Dart-enabled version of Chrome.[a] As we will see later, there is a Dart package that does this for us.

a. http://www.dartlang.org/dartium/

There is more to be said about loading libraries and including code with Dart once we reach Chapter 10, *Libraries*,on page ?. For now, it is simply nice to note that loading Dart works exactly as we might expect it to work.

HTML Body

As for the body of the HTML, there is nothing new there, but we ought to note the IDs of two elements to which we will be attaching behaviors.

```
<h1>Dart Comics</h1>
<p>Welcome to Dart Comics</p>
<ul id="comics-list"></ul>
<p id="add-comic">
  Add a sweet comic to the collection.
</p>
```

To the #comics-list UL element, we are going to attach the list of comic books in the back-end data store. We will also attach a form handler to the #add-comic paragraph tag. So, let's get started.

1.3 Ajax in Dart

We start our Dart application by loading a couple of Dart libraries with a main() function in scripts/comics.dart.

your_first_dart_app/web/scripts/skel.dart

```
import 'dart:html';
import 'dart:convert';
main() {
  // Do stuff here
}
```

As we will see in Chapter 10, *Libraries*,on page ?, there is a lot of power in those import statements. For now, we can simply think of them as a means for pulling in functionality outside of the core Dart behavior.

All Dart applications use main() as the entry point for execution. Simply writing code and expecting it to run, as we do in JavaScript, will not work here. It

might seem C-like at first, but does it honestly make sense that code lines strewn across any number of source files and HTML will all start executing immediately? The main() entry point is more than convention; it is a best practice enforced by the language.

As for the contents of the main() function, we take it piece by piece. We are retrieving a list of comic books in our collection and using that to populate an element on our page.

We need to identify the DOM element to which the list will attach (#comics-list). Next we need an Ajax call to fill in that DOM element. To accomplish both of those things, our first bit of Dart code might look like the following:

your_first_dart_app/web/scripts/comics.dart

```
main() {
  var list_el = document.query('#comics-list');
  var req = new HttpRequest();
  req.open('get', '/comics');
  req.onLoad.listen((req) {
    var list = JSON.decode(req.target.responseText);
    list_el.innerHtml = graphic_novels_template(list);
  });
  req.send();
}
```

Aside from the obvious omission of the function keyword, this example might be JavaScript code! We will cover more differences in Chapter 3, *Functional Programming in Dart*,on page ?. Still in Dart are the semicolons and curly braces that we know and love—the language designers have certainly made the language at least superficially familiar.

Note

Unlike in JavaScript, semicolons are *not* optional in Dart.

In addition to being familiar, this code is easy to read and understand at a glance. There are no weird, legacy DOM methods. We use document.query() for an element rather than document.findByElementId(). And we use the familiar CSS selector of #comics-list, just as we have grown accustomed to in jQuery.

Also note that we are not creating an XMLHttpRequest object. In Dart, it is just HttpRequest. This may seem a trivial change, but remember Dart is written for today's web, not to support the legacy of the web. And when was the last time anyone sent XML over web services?

• Click HERE to purchase this book now. discuss

So far we have the UL that we want to populate and an HttpRequest object to do so. Let's make the request and, after a successful response, populate that UL. As in JavaScript, we open the request to the appropriate resource (/comics), listen for an event that fires when the request loads, and finally send the request.

```
main() {
  var list_el = document.query('#comics-list');
  var req = new HttpRequest();
  req.open('get', '/comics');
  req.onLoad.listen((req) {
    var list = JSON.decode(req.target.responseText);
    list_el.innerHtml = graphic_novels_template(list);
  });
  req.send();
}
```

Most of that code should be immediately familiar to anyone who has done Ajax coding in the past. We open by creating an XHR object and close by specifying the resource to be retrieved and actually sending the request.

It is when we add event handlers that we see a more fundamental departure from the JavaScript way. The XHR object (er, HR object?) has an onLoad property. The onLoad property is a *stream*. Streams are used everywhere in Dart (server-side, client-side, everywhere) as a means of allowing code to receive data without blocking any other code from executing. In this case, the UI should remain responsive until the data from the HttpRequest is available, at which point we do something with it.

In this case, we parse (well, "decode" in Dart) the supplied JSON into a list of hashes, which might look like this:

your_first_dart_app/comics.json

```
[
  {"title":"Watchmen",
   "author":"Alan Moore",
   "id":1},
  {"title":"V for Vendetta",
   "author":"Alan Moore",
   "id":2},
  {"title":"Sandman",
   "author":"Neil Gaiman",
   "id":3}
]
```

With that, we hit the final piece of our simple Dart application—a template for populating the list of comic books.

```
graphic_novels_template(list) {
  var html = '';
  list.forEach((graphic_novel) {
    html += graphic_novel_template(graphic_novel);
  });
  return html;
}
graphic_novel_template(graphic_novel) {
  return '''
    <li id="${graphic_novel['id']}">
      ${graphic_novel['title']}
      <a href="#" class="delete">[delete]</a>
    </li>''';
}
```

The first function simply iterates over our list of comic books (internally, we hipsters think of them as graphic novels), building up an HTML string.

The second function demonstrates two other Dart features: multiline strings and string interpolation. Multiline strings are identified by three quotes (single or double). Inside the string, we can interpolate values (or even simple expressions) with a dollar sign. For simple variable interpolation, curly braces are optional: `$name` is the same as `${name}`. For more complex interpolation, such as hash lookup, the curly braces are required.

And that's it! We have a fully functional, Ajax-powered web application ready to roll. The assembled code is as follows:

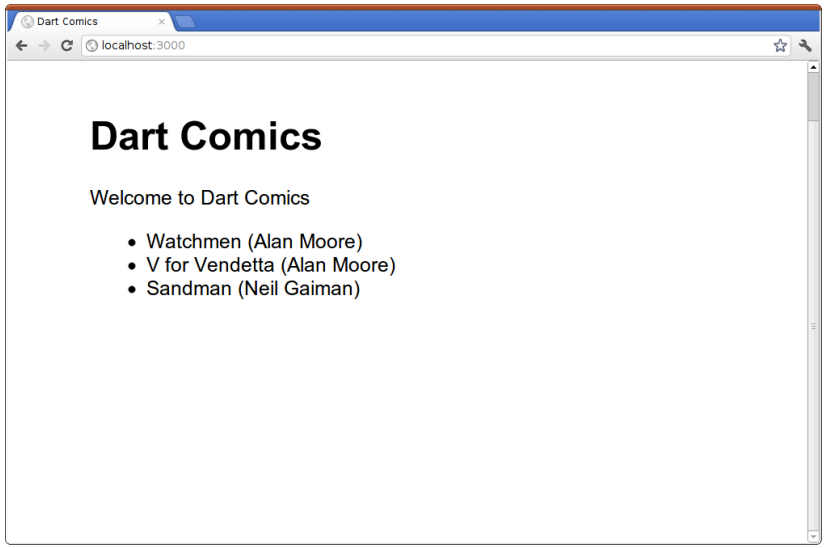
```
import 'dart:html';
import 'dart:convert';

main() {
  var list_el = document.query('#comics-list');
  var req = new HttpRequest();
  req.open('get', '/comics');
  req.onLoad.listen((req) {
    var list = JSON.decode(req.target.responseText);
    list_el.innerHtml = graphic_novels_template(list);
  });
  req.send();
}

graphic_novels_template(list) {
  var html = '';
  list.forEach((graphic_novel) {
    html += graphic_novel_template(graphic_novel);
  });
  return html;
}
graphic_novel_template(graphic_novel) {
```

```
  return '''
    <li id="${graphic_novel['id']}">
      ${graphic_novel['title']}
      <a href="#" class="delete">[delete]</a>
    </li>''';
}
```

And loading the page looks like this:



That is a darned nice start in our exploration of Dart. To be sure, we glossed over a lot of what makes Dart a great language. But in doing so, we have ourselves a very good start on an Ajax-powered web application. Best of all, none of the code that we wrote seemed all that different from JavaScript. Some of the syntax is a little cleaner than what we are used to in JavaScript (no one is going to complain about cleaner code), and those strings are quite nice. But, all in all, it is safe to say that we can be productive with Dart in relatively short order.

1.4 This App Won't Run

As written, this application will not actually work anywhere...well, almost anywhere.

Dart is not supported in any browser (not even Chrome). To run this web application natively, we would need to install Dartium—a branch of Chrome that embeds the Dart VM. Dartium is available from the Dart Lang site.[1]

1. http://www.dartlang.org/dartium/

Even after Dart makes it into Chrome proper, we would still be faced with supporting only a subset of browsers on the market. That is just silly.

Fortunately, Dart can be compiled down to JavaScript, meaning that you can have the power of Dart but still target all platforms. To accomplish that easily, we add a small JavaScript library that, upon detecting a browser that does not support Dart, will load the compiled JavaScript equivalent.

your_first_dart_app/web/index_with_js_fallback.html

```
<!-- Enable fallback to Javascript -->
<script src="/scripts/conditional-dart.js"></script>
```

We will discuss that helper file in detail in Chapter 5, *Compiling to JavaScript*,on page ?. For now, it is enough to note that our Dart code is not locked into a single browser vendor's world. We are very definitely *not* seeing *The Return of VBScript* here.

1.5 What's Next

Admittedly, this has been a whirlwind of an introduction to Dart. It is fantastic to be able to get up and running this quickly. It is even better to feel as though we can be productive at this point.

Still, we are only getting started with Dart, and, make no mistake, our Dart code can be improved. So, let's use the next few chapters to get comfortable with some important concepts in Dart. After that, we will be ready to convert our Dart application into an MVC approach in Chapter 6, *Project: MVC in Dart*,on page ?.

• Click HERE to purchase this book now. discuss