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

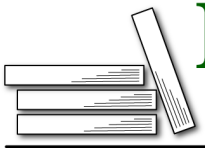
Chris Strom

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 5

Compiling to JavaScript

When Dart first came out, every major browser vendor, as well as the WebKit project, announced that they had absolutely no intention of embedding the Dart VM in their browsers. Many bristled at the very suggestion that a non-standard language be supported even obliquely. How another language was supposed to become a standard seemed a tricky question. Fortunately, Google had a plan.

In the grand tradition of CoffeeScript,[1] the Dart project includes a compiler capable of compiling Dart into JavaScript. The goal is that, even if Dart does not become an overnight standard, web developers tired of the quirks of JavaScript have a choice. We can now code in a modern language for the Web but still support the wide variety of browsers on the market.

The JavaScript generated by the Dart compiler includes shim code for the various Dart libraries. There is generated JavaScript that translates Dart DOM calls into JavaScript. There is generated JavaScript that supports Dart Ajax. For just about every feature of Dart, there is a corresponding chunk of JavaScript that gets included in the compiled output.

If that sounds large, well, it is. When first released, the compiler generated tens of thousands of lines of JavaScript!

Of course, the compiler continues to improve. It now supports compression/optimization and is producing JavaScript libraries in the range of thousands of lines of code instead of tens of thousands. Considering that Dart does a good chunk of the work of many JavaScript libraries like jQuery, this is already a good start.

And it is only going to get better.

1. http://coffeescript.org/

• Click HERE to purchase this book now. discuss

5.1 Compiling to JavaScript with dart2js

The tool provided to compile Dart down into JavaScript is dart2js. The dart2js compiler can be found among the Dart software development kit builds.[2] The SDK are the ones with "sdk" in the filename (as opposed to the "editor" builds that include the editor in addition to the SDK).

Ubuntu hipsters (truly the best hipsters) would use dartsdk-linux-64.zip.

Once unzipped, we see that the SDK contains the entire Dart library (core, html, io, json).

```
.
+-- bin
    +-- dart
    +-- dart2js
    +-- dartanalyzer
    +-- dartdoc
    +-- pub
+-- include
+-- lib
|   +-- async
|   +-- chrome
|   +-- collection
|   +-- convert
|   +-- core
|   +-- html
|   +-- indexed_db
|   +-- io
|   +-- isolate
|   +-- js
|   +-- json
|   +-- math
|   +-- mirrors
|   +-- svg
|   +-- typed_data
|   +-- utf
|   +-- web_audio
|   +-- web_gl
|   +-- web_sql
+-- util
```

In addition to the core Dart libraries, the SDK also contains library code to support documentation (dartdoc) and compiling to JavaScript (dart2js).

2. http://gsdview.appspot.com/dart-editor-archive-continuous/latest/

Using dart2js could not be more basic. It takes a single command line argument—the Dart filename. There is also a single command-line option that can be used to set the filename of the resulting JavaScript (out.js by default).

```
$ dart2js -omain.dart.js main.dart
```

There is no output from the compiler indicating success, but we now have a nice little JavaScript version of our script.

```
$ ls -lh
-rw-r--r-- 1 chris chris   33 Feb 17 12:47 main.dart
-rw-r--r-- 1 chris chris 7.2K Feb 17 12:47 main.dart.js
```

Well, maybe it's not "little."

If there are errors in the Dart code being compiled, dart2js does a very nice job of letting us know where the errors occur.

```
$ dart2js main.dart
main.dart:5:3: Warning: Cannot resolve "document".
  document.query('#foo');
  ^^^^^^^^
```

One thing to bear in mind when compiling JavaScript is that dart2js works only at the application level, not the class level. Consider the situation in which we are converting our comic book collection application to follow a hip MVC pattern.

```
comics.dart
Collection.Comics.dart
HipsterModel.dart
Models.ComicBook.dart
Views.AddComic.dart
Views.AddComicForm.dart
Views.ComicsCollection.dart
```

There is no way to compile individual classes into usable JavaScript.

```
$ dart2js Models.ComicBook.dart
Models.ComicBook.dart:1:1: Error: Could not find "main".
library models_comic_book;

Error: Compilation failed.
```

If the script containing the main() entry point references the other libraries or if those libraries reference other libraries, then everything will be slurped into the resulting JavaScript. The three libraries referenced in the following import statements will be pulled into the compiled JavaScript:

compiling_javascript/web/scripts/comics.dart
```
import 'Collections.Comics.dart' as Collections;
import 'Views.Comics.dart' as Views;

main() {
  // ...

}
```

Similarly, the ComicBook model will also be included in the dart2js-generated JavaScript by virtue of being referenced in the collection class.

compiling_javascript/web/scripts/Collections.Comics.dart
```
library comics_collection;

import 'HipsterCollection.dart';
import 'Models.ComicBook.dart';

class Comics extends HipsterCollection {
  // ...

}
```

At some point, it might be nice to write classes in Dart and compile them into usable JavaScript. For now, however, we are relegated to compiling entire applications, not pieces.

5.2 Maintaining Dart and JavaScript Side-by-Side

As Dart and dart2js evolve, the performance of the generated JavaScript will improve. Even at this early stage, Dart code compiled to JavaScript rivals and sometimes surpasses what the typical JavaScripter might write[3]. But as fast as the compiled JavaScript gets, it will never be as fast as running Dart natively.

The question then becomes, how can we send Dart code to Dart-enabled browsers and send the compiled JavaScript to other browsers?

The answer is relatively simple: include a small JavaScript snippet that detects the absence of Dart and loads the corresponding JavaScript. As we saw in the previous section, if we compile a main.dart script, then dart2js will produce a corresponding main.dart.js JavaScript version.

The following JavaScript snippet will do the trick:

3. See the Dart performance page for details on how that is determined: https://www.dart-lang.org/performance/.

```
if (!navigator.webkitStartDart) loadJsEquivalentScripts();

function loadJsEquivalentScripts() {
  var scripts = document.getElementsByTagName('script');
  for (var i=0; i<scripts.length; i++) {
    loadJsEquivalent(scripts[i]);
  }
}

function loadJsEquivalent(script) {
  if (!script.hasAttribute('src')) return;
  if (!script.hasAttribute('type')) return;
  if (script.getAttribute('type') != 'application/dart') return;

  var js_script = document.createElement('script');
  js_script.setAttribute('src', "${script.getAttribute('src')}.js");
  document.body.appendChild(js_script);
}
```

There is a similar script in Dart core.[4] In most cases, that script should be preferred over ours because it does other things (such as start the Dart engine).

The check for an available Dart engine is quite simple—if the browser knows how to start the Dart engine, then it is Dart-enabled.

```
if (navigator.webkitStartDart)
```

That may come in handy elsewhere in our Dart adventures.

The remainder of the JavaScript is fairly simple. The loadJsEquivalentScripts() function invokes loadJsEquivalent() for every <script> tag in the DOM. This method has a few guard clauses to ensure that a Dart script is in play. It then appends a new .js <script> to the DOM to trigger the equivalent JavaScript load.

To use that JavaScript detection script, we save it as dart.js and add it to the web page containing the Dart <script> tag.

```
<script src="/scripts/dart.js"></script>

<script src="/scripts/main.dart"
          type="application/dart"></script>
```

A Dart-enabled browser will evaluate and execute main.dart directly. Other browsers will ignore the unknown "application/dart" type and instead execute the

4. It is part of the browser Pub package available at: http://pub.dartlang.org/packages/browser. We will talk more about Pub and library packages in Chapter 10, *Libraries*,on page ?

code in dart.js, creating new <script> tags that source the main.dart.js file that we compiled with dart2js.

In the end, we have superfast code for browsers that support Dart. For both Dart and non-Dart browsers, we have elegant, structured, modern code. Even this early in Dart's evolution, we get the best of both worlds.

5.3 What's Next

The ability to compile Dart into JavaScript means that we do not have to wait for a tipping point of browser support before enjoying the power of Dart. Today, we can start writing web applications in Dart and expect that they will work for everyone. This is a good thing because, in the next chapters, we will be taking our simple Dart application to the next level and we wouldn't want to leave our nonhipster friends too far behind.

• Click HERE to purchase this book now. discuss