A New Language for Building Structured Web Apps

What is Dart?

Kathy Walrath & Seth Ladd

Master the Web's Most Important Technologies

JavaScript & Beyond

MAY 29 – 31, 2012
SAN FRANCISCO, CA

Explore the Changing Worlds of JavaScript and HTML5

Make sense of the vast explosion of JavaScript and related technologies and take away practical skills you can apply immediately at the **O'Reilly Fluent Conference**.

Conference tracks include: Ancillary Technologies, JavaScript in the Browser, Mobile, Pure Languages, Server Side, and Stack Track.

Fluent Conference is for JavaScript developers, web developers, web performance engineers, and mobile app developers.

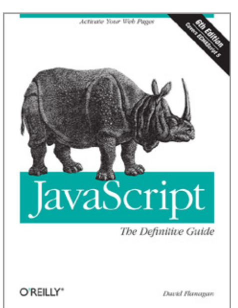
Registration is now open at fluentconf.com

Save 20% with code **REPORT20**

©2012 O'Reilly Media, Inc.
O'Reilly logo is a registered trademark of O'Reilly Media, Inc. 12256

O'REILLY®

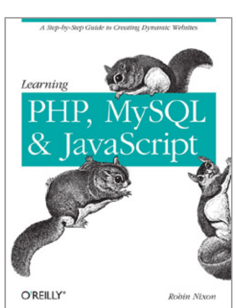
Related Ebooks



JavaScript: The Definitive Guide, 6th edition

By David Flanagan
Released: April 2011
Ebook: **$39.99**

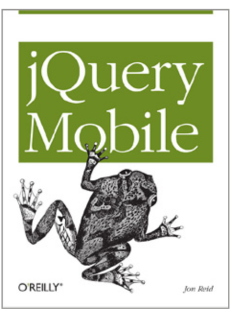
Buy Now



Learning PHP, MySQL, and JavaScript

By Robin Nixon
Released: July 2009
Ebook: **$31.99**

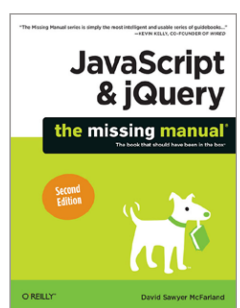
Buy Now



jQuery Mobile

By Jon Reid
Released: June 2011
Ebook: **$12.99**

Buy Now



JavaScript & jQuery: The Missing Manual

By David Sawyer McFarland
Released: October 2011
Ebook: **$31.99**

Buy Now

What is Dart?

Kathy Walrath and Seth Ladd

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Download from Wow! eBook <www.wowebook.com>

What is Dart?
by Kathy Walrath and Seth Ladd

Copyright © 2012 O'Reilly Media, Inc. All rights reserved.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

March 2012: First Edition.

Revision History for the First Edition:

2012-03-07 First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449332327* for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *What is Dart?* and related trade dress are trademarks of O'Reilly Media, Inc.

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 O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-33232-7
1331845355

Table of Contents

What is Dart? **1**
Why Did Google Create Dart? 1
Does the Web Really Need Another Language? 2
Show Me the Code 2
How Can I Play with Dart? 3
How About a Real Editor? 5
What's New About Dart? 6
Why Does Dart Look so Familiar? 8
What Is in the Dart Platform? 8
Should I Use Dart for My App Today? 8
How Do You Expect People to Use Dart? 9
How Can I Compile to JavaScript? 9
What Libraries Are Available? 10
dart:core 10
dart:html 10
dart:io 11
Show Me More Code 11
Types 11
Generics 12
Manipulating the DOM 13
Isolates 14
Where Can I Learn More? 15

What is Dart?

Dart is a new language developed by Google that's getting attention in web app circles. We asked Kathy Walrath and Seth Ladd, members of Google's developer relations team, to explain Dart's purpose and its applications.

Writing a web app can be lots of fun, especially at the beginning when you experience instant gratification: code, reload, repeat.

Unfortunately, finishing and maintaining a web app are not so fun. JavaScript is great for small scripts, and it has the performance chops to run large apps. But when a script evolves into a large web app, debugging and modifying that app can be a nightmare, especially when you have a large team.

Enter Dart, an open-source project that aims to enable developers to build more complex, highly performant apps for the modern web. Using the Dart language, you can quickly write prototypes that evolve rapidly, and you also have access to advanced tools, reliable libraries, and good software engineering techniques.

Even though Dart is young, it already has tools such as **Dartboard** (which lets you write and run Dart code in your browser) and **Dart Editor** (which lets you create, modify, and run Dart apps). A recently released **SDK** contains command-line tools such as a **Dart-to-JavaScript compiler** (which produces JavaScript that you can put in any modern browser) and a **Dart Virtual Machine** (the **VM**, which lets you run Dart code on servers). The latest tool to become available is a build of the Chromium browser, nicknamed **Dartium**, that contains a built-in Dart VM.

*(**Note: Dart is still changing.** This article is correct as of March 2012, but facts might change and links might go bad. For the latest information, see the Dart website.)*

Why Did Google Create Dart?

We want you to be able to create great web apps. Great web apps improve the web, and when the web does better, everyone wins.

Engineers at Google have been thinking about web apps for a long time. We've written a bunch of complex and widely used large-scale web apps (think Gmail, Google+, and

Google Docs), so we're quite familiar with the challenges of architecting web apps. We've also written a browser (Chrome) and a JavaScript engine (V8), so we've thought a lot about how to make web apps run faster.

Basically, we created Dart because we think it'll help bring more great apps to the web, and we think it should be easier to create more complex web applications.

Does the Web Really Need Another Language?

App developers from all platforms should be able to build for the modern web, but many non-endemic web developers have different expectations and requirements from their language, tools, and development platform than what is available today. We believe there is room for a new platform, one that is not encumbered by 15 years of cruft, that is familiar to developers of different backgrounds, and that is structured to enable the larger, more complex apps that users are demanding.

We don't think JavaScript is going away. In fact, Google is actively working with TC39 to improve JavaScript, and new features are already landing in V8 and Chrome. However, our commitment to improving JavaScript doesn't prevent us from thinking about other solutions. For example, Dart will take advantage of the continued work on JavaScript, because Dart programs compile to JavaScript to run across the entire modern web.

We believe Dart is a compelling and familiar platform that meets the needs of developers from different backgrounds and experiences, including endemic web developers. We believe that Dart brings fresh ideas to web programming, and that innovation in both Dart and JavaScript will help push the web forward for app developers and users.

Show Me the Code

Enough talk, let's see some code. Whether you know JavaScript or Java, Dart code should look familiar. Here's an example of a simple web page in Dart:

hi.dart:

```
#import('dart:html');

main() {
  document.query('#status').text = 'Hi, Dart';
}
```

hi.html:

```
...
<h2 id="status"></h2>
<script type="application/dart" src="hi.dart"></script>

<!--
  If the browser doesn't have an embedded Dart VM,
```


Download from Wow! eBook <www.wowebook.com>

```
  you can compile Dart code to JavaScript.
-->
<script type="text/javascript" src="hi.dart.js"></script>
...
```

Now let's look at some Dart code that uses functions:

```
send(msg, to, from, [rate='First Class']) {
  return '${from} said ${msg} to ${to} via ${rate}';
}

main() => print(send('hello', 'Seth', 'Bob'));

> "Bob said hello to Seth via First Class"
```

The `=>` syntax used to implement main() is a nice, compact way to implement a function that evaluates and returns a single expression. Without `=>`, the implementation of the main() method would look like this:

```
main() {
  print(send('hello', 'Seth', 'Bob'));
}
```

In the send() method implementation above, *rate* is an optional parameter with a default value. That method also illustrates string interpolation at work (`${var}`).

Here's some Dart code that's more object-oriented:

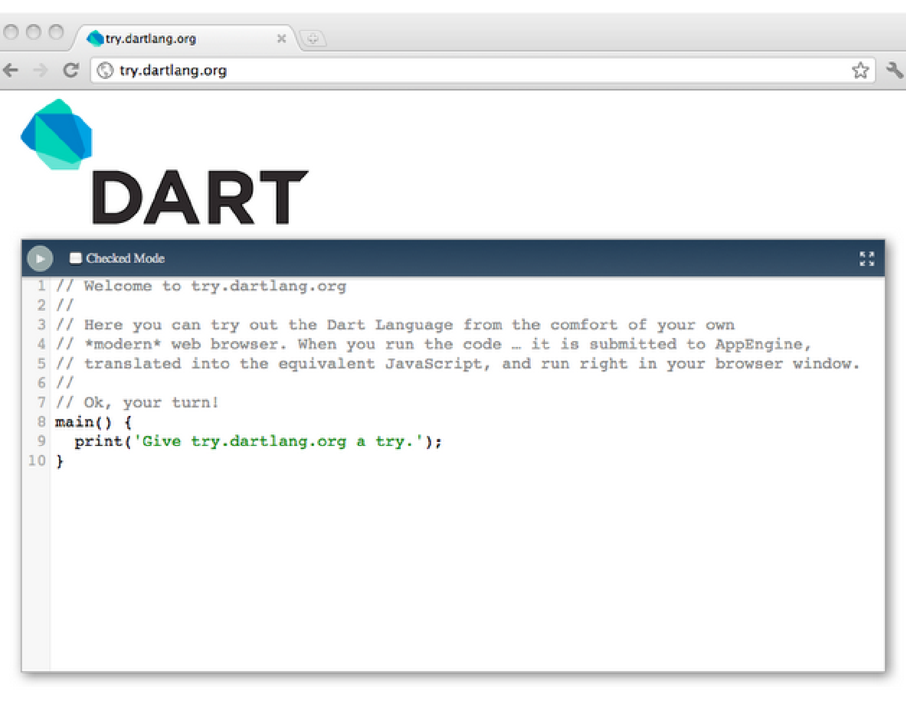
```
class Point {
  Point(this.x, this.y);
  distanceTo(other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
  var x, y;
}

main() {
  var p = new Point(2, 3);
  var q = new Point(3, 4);
  print('distance from p to q = ${p.distanceTo(q)}');
}
```

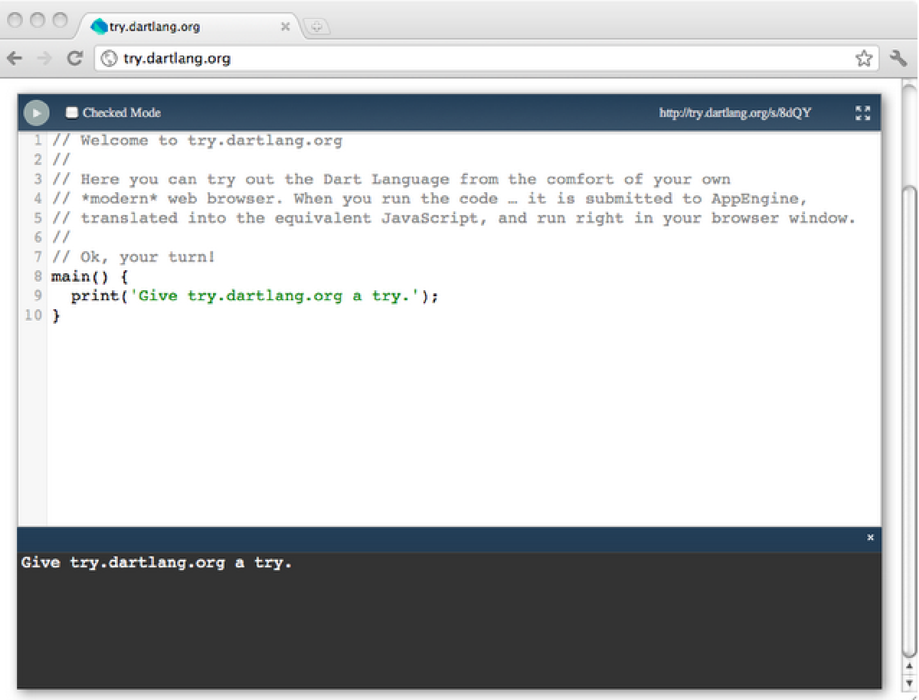
This code should look pretty familiar if you've ever used a class-based language.

How Can I Play with Dart?

The easiest way to try out Dart is to use Dartboard, a way to execute Dart code interactively in any modern browser. Dartboard is embedded in the Dart site, at www.dartlang.org. You can also use Dartboard by going directly to try.dartlang.org. Here's a picture of what you'll see on that site:



Change the code as you like, and then click the Run button (at the upper left). You'll see the results of your code in a new area below:

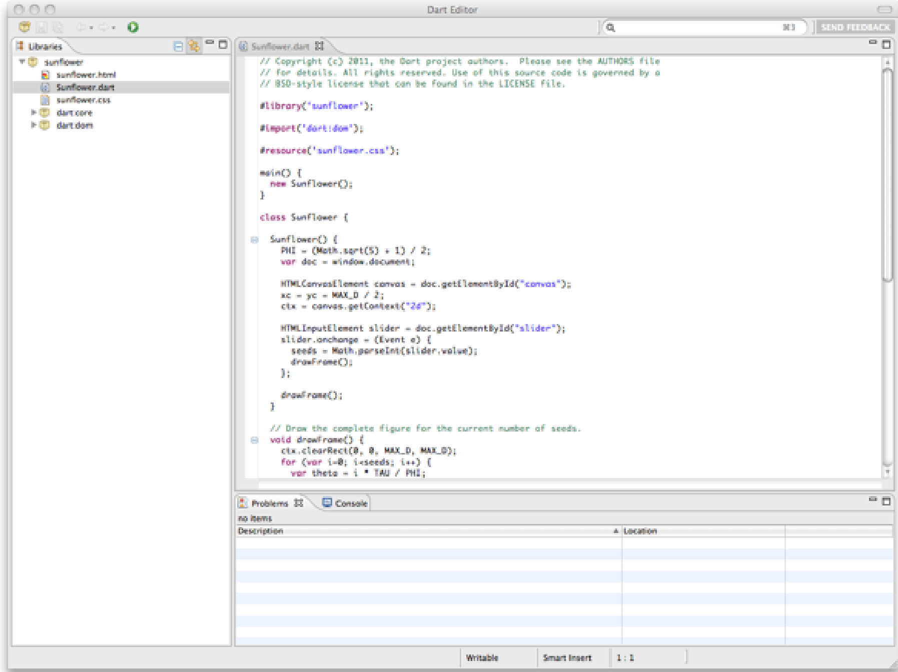


The URL at the upper right of the Dartboard is a link to the code you just ran.

Like most things related to Dart, Dartboard is still changing. For the latest information, see the Dartboard tutorial.

How About a Real Editor?

When you outgrow Dartboard, try Dart Editor. It's a downloadable editor for Windows, Mac, and Linux that lets you write, modify, and run Dart web apps. Dart Editor can run your Dart code via the VM, or it can compile it to JavaScript and launch it in your browser. Here's a picture of what Dart Editor currently looks like:



To run any program in Dart Editor, just click the Run button while any item in that program's library is selected. If your program is a web app, Dart Editor launches Dartium (Chromium with an embedded Dart VM) to run the program. When you are ready to ship to production, the Dart Editor can compile it to JavaScript, making your app available to the entire modern web.

Dart Editor has several features to help you edit Dart code, and we expect more features soon. As you can see from the screenshot above, Dart Editor highlights Dart syntax. Dart Editor also supports auto-completion, and it can quickly take you to where types and other APIs are declared. You can also get a quick outline of your program's classes, methods, and fields.

See the Dart Editor tutorial for download instructions and a walkthrough.

What's New About Dart?

Now that you're all tooled up, let's talk about the language. We're not going to go into Dart's features and syntax in detail—you can read about all of that on www.dartlang.org—but here are a few of the more interesting features of Dart.

Optional typing: You can use types or not, it's up to you. Types in Dart code don't change the way your app executes, but they can help developers and programming

tools to understand your code. You might not bother with types while you're developing a prototype, but you might add types when you're ready to commit to an implementation. An emerging pattern is to add types to interfaces and method signatures, and omit types inside methods.

Snapshots: Currently, browsers need to parse a web app's source code before that app can run. Dart code can be snapshotted—all its state and code recorded at a certain point in time—which can speed up startup considerably. In initial testing, a web app with 54,000 lines of Dart code started up in 640ms without snapshotting. With snapshotting, it started up in 60ms. When your Dart program is running in the Dart VM, it can see significant startup time performance improvements, thanks to snapshots.

Isolates: Dart supports concurrent execution by way of *isolates*, which you can think of as processes without the overhead. Each isolate has its own memory and code, which can't be affected by any other isolate. The only way an isolate can communicate with another isolate is by way of messages. Isolates allow a single app to use multi-core computers effectively. Another use for isolates: running code from different origins in the same page, without compromising security. For more details see "Isolates" on page 14, below.

Interfaces with default implementations: Ignore this one if you've never used a language that features classes and interfaces/protocols. Still here? OK. If you take a look at the Dart libraries, you'll notice that they use interfaces in cases where some other languages would use classes—for example, for Date and HashMap. This is possible because a Dart interface can have a default implementation—a class (usually private) that is the default way to create objects that implement the interface. For example, even though Stopwatch is an interface, you can call `new Stopwatch()` to get a default stopwatch implementation. If you look at the API doc or source code, you can see that the default implementation of Stopwatch is a class named StopwatchImplementation.

Generics, but easy: Generics have been done before, but they've been confusing. Dart takes a new approach by designing a generics system that's more understandable. The tradeoff is sacrificing a bit of exactness, but we believe that enabling developers to be more productive trumps ivory-tower language design. For more details see "Generics" on page 12, below.

HTML library: We also took a fresh look at how you should use the HTML DOM. (*DOM* is short for *Document Object Model*; it's the interface that lets you programmatically update the content, structure, and style of a web page.) By creating a native Dart library (dart:html) to access and manipulate the DOM, we made elements, attributes, and nodes feel natural to work with. More details are in the sections "What Libraries Are Available?" on page 10 and "Manipulating the DOM" on page 13.

Why Does Dart Look so Familiar?

A cutting edge and unique language might be beautiful, but it would have maybe five users. Dart is designed for mass adoption, so it has to feel familiar to both scripting language users (such as JavaScripters) and structured language users (such as Java developers).

Still, Dart has some unique features for a language targeted at the mainstream. For example, interfaces with default implementations help with hiding implementation details. Optional typing, another new feature for the web, should help with more clearly annotating developer intention and interface and library contracts.

What Is in the Dart Platform?

Dart is more than just a language, it's an entire platform for modern web developers.

Language specification: The Dart language is familiar, with a few new features such as optional typing and isolates.

Libraries: Core libraries provide functionality including collections, dates, and math, as well as HTML bindings, server-side I/O such as sockets, and even JSON.

Compiler to JavaScript: You can compile Dart programs to JavaScript that can run across the entire modern web.

VM: The virtual machine is built from the ground up to run Dart code natively. The VM runs on the command line for server-side applications, and can also be embedded into browsers for client-side applications.

Integration with Chromium: The Dart VM has been embedded into a build of Chromium, nicknamed Dartium, allowing native Dart applications to run without first being compiled to JavaScript.

Dart Editor: This lightweight editor, complete with syntax highlighting and code completion, can launch your script in the VM or web app in Dartium. It can also compile your web app to JavaScript and run it in another browser.

Should I Use Dart for My App Today?

Dart is still changing, so for now, don't bet the farm on it. However, everything in Dart is open source and open to debate, so we encourage you to check out Dart and provide feedback. Once the language and libraries settle down (this year, if all goes according to plan) we expect that you'll be able to run production Dart web apps in all major modern browsers.

Download from Wow! eBook <www.wowebook.com>

How Do You Expect People to Use Dart?

You can use Dart to build complex, high-performance apps for the modern web. The Dart language is designed to work on the client and the server, which means that you can use it to implement a complete, end-to-end application.

You can also use Dart to scale your development as your program grows from a small set of functions to a large collection of classes. For instance, web development encourages extremely short development iterations. As you refine your idea and your program grows in scope and complexity, your code will probably need more modularity and encapsulation. Dart enables you to refactor from functions to classes, and from untyped code to typed code.

As for deployment, you have two options: compiling to JavaScript or using the Dart VM. For your client-side code, compiling to JavaScript enables your Dart code to run on modern browsers. A future version of Chrome will ship with an embedded Dart VM, allowing your Dart code to run directly in Chrome without first being compiled to JavaScript.

How Can I Compile to JavaScript?

To produce human readable JavaScript, you should use the Frog compiler (available in the Dart SDK). Still a work in progress, Frog generates JavaScript code that is small and performant.

For example, say you have the following Dart code:

```
class HelloWorld {
  yell() => print("hello world!!!!");
}

main() => new HelloWorld().yell();
```

Frog compiles that into the following JavaScript code:

```
//  ********** Library dart:core **************
//  ********** Natives dart:core **************
// ********** Code for Object **************
// ********** Code for top level **************
function print(obj) {
  return _print(obj);
}
function _print(obj) {
  if (typeof console == 'object') {
    if (obj) obj = obj.toString();
    console.log(obj);
  } else {
    write(obj);
    write('\n');
  }
}
```

```
//  ********** Library dart:coreimpl **************
// ********** Code for NumImplementation **************
NumImplementation = Number;
// ********** Code for StringImplementation **************
StringImplementation = String;
// ********** Code for _Worker **************
// ********** Code for top level **************
// ********** Library helloworld **************
// ********** Code for HelloWorld **************
function HelloWorld() {

}
HelloWorld.prototype.yell = function() {
  return print("hello world!!!!");
}
// ********** Code for top level **************
function main() {
  return new HelloWorld().yell();
}
main();
```

As you can see, the JavaScript code is straightforward.

To get the Frog compiler, download the SDK, which is available now in prerelease.

What Libraries Are Available?

For now, Dart has three libraries that you can use:

dart:core

The basic APIs that all apps can count on, whether they're standalone scripts or inside the browser. These APIs let you perform operations such as:

- display basic text (print())
- group objects in collections (Collection, Set, List, Queue)
- manipulate key-value pairs (Map)
- specify dates and times (Date, TimeZone, Duration, Stopwatch)
- use strings (String, Pattern, RegExp)
- use numbers (num, int, double, Math)
- return a value before your operation is complete (Future)
- compare similar objects (Comparable)
- perform an operation on each item in a multi-item object (Iterable)

dart:html

APIs for producing UIs for web apps. For example:

- get global objects (document, window)
- find HTML elements (Element's query() and queryAll())
- add and remove event handlers (Element's on property)
- operate on groups of objects using the built-in Dart collection interfaces

For an example of using this library, see the "Manipulating the DOM" on page 13 section.

dart:io

Server-side APIs for connecting to the outside world. This library is accessible only when the Dart VM is running on the server. Examples:

- read and write data (InputStream, OutputStream)
- open and read files (File, Directory)
- connect to network sockets (Socket)

You can see the API documentation at api.dartlang.org.

The community has also started porting libraries that you can use. For example, the crypto libraries from Closure have been ported to Dart. For game development, the popular Box2D physics engine has also been ported to Dart.

Show Me More Code

We've given an overview of Dart's most interesting features and shown some Dart code. Now we'll elaborate on a few of these features—types, generics, isolates, and DOM manipulation—and show yet more code.

Types

Up above, we showed this example:

```
class Point {
  Point(this.x, this.y);
  distanceTo(other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
  var x, y;
}

main() {
  var p = new Point(2, 3);
  var q = new Point(3, 4);
  print('distance from p to q = ${p.distanceTo(q)}');
}
```

You might notice that although the code defines and uses a class, it otherwise has no types. Types are optional in Dart. They don't change the way programs run, but they make the code more understandable by tools (such as debuggers and IDEs) and developers (such as your replacement, when you move on to an even better project). Think of types as annotations or documentation that can help tools and humans understand your intention better and catch bugs earlier.

We think a good place to start adding types is method signatures and interfaces—the "surface area" of your program. As more people are added to your project and more classes are generated, it's useful to know what types are used by (and returned from) methods.

Here's the previous sample with types added to the API definitions:

```
class Point {
  Point(this.x, this.y);
  num distanceTo(Point other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
  num x, y;
}

main() {
  var p = new Point(2, 3);
  var q = new Point(3, 4);
  print('distance from p to q = ${p.distanceTo(q)}');
}
```

Notice how we didn't add types to the method body of distanceTo(). An emerging pattern of idiomatic Dart is to use types for method signatures but not method bodies. Method bodies should be small enough to be easily understood, and we expect tools to perform type inference at some point. However, if you prefer to use types everywhere, you're free to change `var dx` to `num dx`, and so on.

Generics

If you haven't used generic types before—or maybe even if you have—seeing something like List<E> in the API reference might be a bit scary. Don't worry. Generics in Dart are easy peasy.

For example, if you don't care what types of objects are in a List, then you can create a list like this:

```
new List()
```

If you know that your list will only have one kind of object in it—only strings, for example—then you can (but don't have to) declare that when you create the List object:

```
new List<String>()
```


Download from Wow! eBook <www.wowebook.com>

Why bother with the extra ceremony? Specifying what types your collection can hold is a good way to document to both your fellow programmers and your tools what your expectations are. The tools and the runtime can detect bugs early on.

(**Fact:** `new List()` is shorthand for `new List<Dynamic>()`. `Dynamic` is the type used behind the scenes for "untyped" variables.)

Here are examples of how Dart treats different types and untyped collections. If you're familiar with generics in Java, pay close attention to the last two examples.

```
main() {
  print(new List() is List<Object>);
  print(new List() is List<Dynamic>);
  print(new List<String>() is List<Object>);
  print(new List<Object>() is! List<String>); // Not all objects are strings
  print(new List<String>() is! List<int>);    // Strings are not ints
  print(new List<String>() is List);          // TRUE! Every list of string is
                                              // a list
  print(new List() is List<String>);          // TRUE! It's OK to pass a List
                                              // to a method that expects
                                              // List<String>
}
```

The last two examples highlight Dart's *covariant* generics. Because you will run into untyped Dart code in the wild, the Dart language must allow you to treat an untyped List the same as a typed List, and vice versa.

As you may be able to tell from the code above, Dart's parameterized types are *reified*. This means that the generics aren't lost at compile time, so Dart truly knows that a `List<String>` is a "list of Strings".

For more tips on using generics, see the Generics section of the Optional Types in Dart article. Also see the blog post Generics in Dart, or, Why a JavaScript programmer should care about types.

Manipulating the DOM

Revisiting the DOM to make it more Dart-esque has a lot of advantages. You can use elements and nodes as actual Dart objects, and you can loop through child nodes just like you can with other Dart collections.

For example, here's some code that finds a specific element and then performs various operations on the element and its children.

```
#import("dart:html");

main() {
  // Find an element.
  var elem = document.query("#id");

  // Handle events.
  elem.on.click.add((event) => print('click!'));
```

```
// Set an attribute.
elem.attributes['name'] = 'value';

// Add a child element.
elem.elements.add(new Element.tag("p"));

// Add a CSS class to each child element.
elem.elements.forEach((e) => e.classes.add("important"));
}
```

For more information, read the article Improving the DOM, and browse the dart:html library.

Isolates

Even though Dart is single threaded, you can still take advantage of multi-core machines by using isolates. An isolate provides memory isolation between different parts of a running program. Each isolate can run in a separate thread or process, managed by the Dart VM.

Isolates communicate by sending messages through ports. The messages are copied so that an isolate can't change the state of objects that belong to other isolates.

To illustrate isolates at work, let's build a simple echo service. First we will define echo(), a function to run in an isolate. Each isolate has a port, which we can use to receive messages. We'll reply to the message via the provided reply port.

The main() method creates a new isolate for the echo() function with spawnFunction(). Use the SendPort to send messages to the isolate, and listen for any responses using then().

```
#import('dart:isolate');

    echo() {
    port.receive((msg, reply) => reply.send("Echo: $msg"));
    }

    void main() {
    SendPort sendPort = spawnFunction(echo);
    sendPort.call("Hello, Dart!").then((response) => print(response));
    }
```

Here's the output:

```
Echo: Hello, Dart!
```

We tested the preceding code on the Dart VM. In JavaScript, isolates compile to web workers so that they can run in separate processes. Isolates are still getting refactored, stay tuned for more updates.

Where Can I Learn More?

The main place to go for Dart information is dartlang.org.

Here are some places to get news about Dart:

- Official news: Dart News & Updates
- Google+:
  - —Dart: Structured web apps (unofficial Google+ page)
  - —#dartlang (hashtag to find on Google+ posts)
- Twitter:
  - —@dart_lang (official Dart tweets)
  - —#dartlang (hashtag to find Dart tweets; also see #dart)
- Blogs: Dartosphere (single feed of many Dart blogs)

You can discuss Dart at our main group.

Other official groups give you access to detailed information such as code reviews, commits to the source code repository, and issue tracker changes:

If you want to see how Dart is implemented or you want to become a contributor, check out the Dart open-source project.

And finally, here are some good interviews and videos about Dart:

- Transcription of Gilad Bracha's quick tour of Dart (2 Nov 2011)
- InfoWorld interview with Lars Bak (15 Nov 2011)
- Programming Dart: Chromium + Dart + Dev Tools + Breakpoints (24 Jan 2012)
- Dartisans hangout with Seth Ladd, Vijay Menon, and Anton Muhin (15 Feb 2012)

Dart is getting ready. We encourage you to learn more about Dart, play with Dart in your browser, browse the API docs, enter feature requests and bugs, and join the discussion. Please give it a try. We look forward to your feedback.

Download from Wow! eBook <www.wowebook.com>