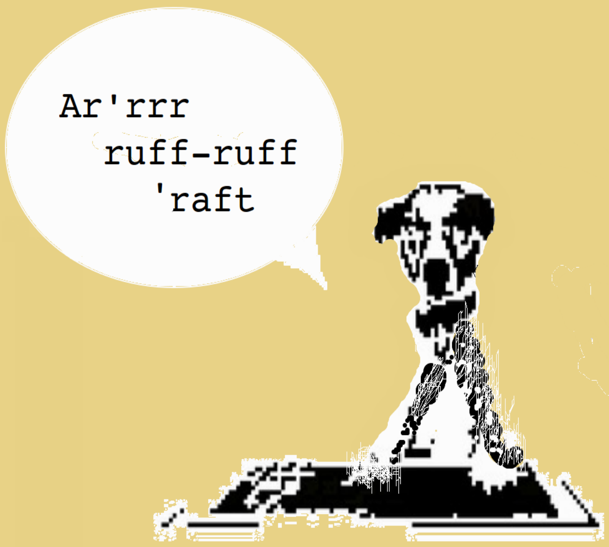
Google's Spark

An Integrated Development Enviroment (IDE) for the Dart programming language and Polymer library



Teaching a New Pooch an Old Trick

James Sosontovich

Spark

Dart/Polymer IDE

James Sosontovich

This book is for sale at http://leanpub.com/spark-ide

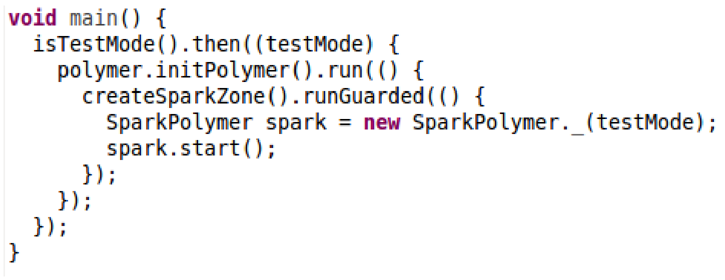
This version was published on 2014-03-31

Contents

Chapter One: The Handwriting on the Wall

I REMEMBER ATTENDING a class where the professor drew a cryptic mathematical expression on the blackboard, stood aside, and then declared that if we trusted in his guidance we would understand its meaning by the end of the course.

He was correct. Of course, it's code we're dealing with here, not mathematical equations. Still I always recall this experience when trying to learn something new.



```
void main() {
  isTestMode().then((testMode) {
    polymer.initPolymer().run(() {
      createSparkZone().runGuarded(() {
        SparkPolymer spark = new SparkPolymer._(testMode);
        spark.start();
      });
    });
  });
}
```

Google's Code on the Blackboard

If you only have an inkling about how this code works then hang on because we're in for a ride.

But before we get to Google's Integrated Development Environment (IDE) allow me to continue along the lines of my professor's introduction.

In the case of the Code on the Blackboard, SparkPolymer spark = new SparkPolymer._(testMode), it inspires the telling of a Biblical story.

Belshazzar, the king of Babylon, while throwing a party for friends and family drank wine from a sacred cup and upset the most Holy God. As a consequence of his prideful deed a hand appeared out of nowhere and began to write words on the wall.

Daniel 5, 1-31, of the Holy Bible gives the details; for brevity I'll stick to the highlights.

The king, upset by this supernatural event, promises a fortune of gold and a powerful position of third highest in the land to whoever interprets the meaning of these words.

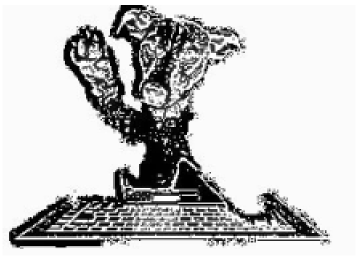
Okay, granted, comparing Google's code example to these strange words written by a detached hand is a stretch of the imagination; however a promise of fortune and power for the *interpreters* is an intriguing prospect.

But here's the catch 22 of the Biblical story. A fellow by the name of Daniel was confident that with the help of God he'll be able to understand the meaning of the handwriting on the wall, yet he didn't want anything in return. An open-source sort of philosophy? Can we trace the roots of this movement to Daniel? Anyhow, he does interpret these words and it's bad news for the king. Even so the king is true to his promise and rewards Daniel with the greatest of treasures and a vast amount of power.

Great story! I recommend you read it for yourself.

Here's the amusing part. My professor told his story to clear away those classmates who weren't interested in understanding his blackboard equation. I tell the story of Daniel to state my position about open-source software. Simply. This book is about an open-source project, Spark, and both, likewise, should be given away openly and under the same software license. Hopefully, like in the case of Daniel, or perhaps concerning the fate of Spark and the Dart programming language and its Polymer library, the rightful blessings will materialize for all those involved just like the *hand* materialized before king Belshazzar ... many moon ago.

Bye-the-way, let me introduce you to our mascot, 'Spark'; that's him on the keyboard. As my editor and chief I can already hear him howling at those moons. As he says, this is going to be 'ar-ruff-ruff' draft.



Spark

Chapter Two: A Brief History

Why?

CERTAINLY there's a sharp critical eye set against yet another programming language. Can you blame this type of attitude from the poor grunt programmers, like myself, just trying to do their humble jobs? What's really different about Dart from its root language of C or its late object oriented cousin language, C++, or the ever popular Java, that would provide such motivation as to create yet another computer language? I'd hazard to guess it's something like the addition of a functional style of programming to Dart, something like JavaScript, or Scala … yup, that would certainly up the complexity of any computer language?

Today we, as programmers, are bombarded by a host of different computer languages. It takes humongous effort just to navigate through this wilderness. All the while our king employers, in this new land of Babylon, wait upon their interpreters to do their jobs. Or perhaps, if we're lucky, our king employers make the decision for us as to which language we use; and yet, somehow, somewhere, we, as programmers, long to connect with something special. That one magical language. Ah, such lofty dreams.

Enter Dart. Or as Spark likes to bark out, D'art. Putting stress on the art side of Dart. Hopefully bridging the gap between the artist in us and the programmer, perhaps?

Of course all of this borders on nonsense. There is no magical computer language.

Dart was designed so that it would feel familiar to those programmers coming from an object oriented background. C++ and Java programmers in particular, however, perhaps Dart's the crossover language for us 'other' 'non-typed' functional 'event-driven' language programmers who never got a chance to work with such structured languages as C++ or Java? Or, perhaps, too afraid of that kind of responsibility?

And what about Spark, our spaghetti loving mascot, tired of picking those long strings of noodles out of his furry coat; what does he say about his newest choice? Yup, let's take a closer look at what Dart has to offer.

What?

DART is a class-based, single inheritance, object-oriented language with C-style syntax. It supports interfaces, abstract classed, reified generics, and optional typing. Static type annotations do not affect the runtime semantics of the code. Instead, the type annotations can provide documentation for tools like static checkers and dynamic run time checks.

On the surface, other than optional typing Dart doesn't sound much different than Java or C++ semantically.

Performance wise, all programmers know that C++ compiled to native code is fast, lightening fast. On the other hand, if your boss isn't in a rush, well, Java is a pretty language, clean and ordered, semantically, runs everywhere, but most bosses have zero tolerance, no patients for waiting on anyone, anything, let alone a slow running computer program. And God forbid you break the budget on buying some decent hardware.

So is Dart the silver bullet of performance? As it stands to date Dart is compiled down into JavaScript and runs under the Chrome's V8 engine or any other compatible browser platform. That still bench-marks around twenty times slower than a C++ program running natively. The interesting part about Dart's future is that Google is working on a new virtual platform to run Dart within its Chrome browser. They have even recruited a gaming guru into their fold and are hoping to push the performance of their Dart language to the max. Imagine a browser application running at C++ speeds? Is that even possible?

Personally, it's a tall order to fill. In the meantime there's a plan Google is following to achieve this goal. First, unlike JavaScript which is a prototype-based language, Dart is based upon static objects. Finding methods and properties in the prototype chain is slow as compared to the efficient lookup of a static object. Second, storing arrays in Javascript is messy since the data can be spread out in memory while Dart has hole free arrays. Third, Dart distinguishes between growable and fixed sized arrays while this is lacking in JavaScript. Fourth, distinction between integer and double means better control over how much memory is employed. Now for the biggie, SIMD? At present machine code, the code which the VM will run, under JavaScript, is being processed as Single Instruction Single Data (SISD). This part of the processor, our true and trusted CPU, is well used; however the newer part of the current CPU that processes Single Instruction Multiple Data (SIMD) is not being taken advantage of; hence the recruitment of the gaming guru.

So here we are following along with Google down at the low level of the machine code where performance gains can be achieved. It's a tall order, yes, but reading about their game plan, it makes one wonder? Hypothetically, by taking advantage of the SIMD processing speed gains it just maybe possible to create a faster GUI which, under the surface, are nothing more than vector processes. Of course I must remind myself that we're talking about an experimental VM; for now, Chrome runs Dart as compiled JavaScript.

Bottom line: Google's technical advances maybe considered a game changer for the software development community. Therefore one big bark from Spark.

Apart from optional typing in Dart, SparkPolymer spark = new SparkPolymer._(testMode), where the variable type SparkPolymer is the sugar in this example, let's just say it's a nice feature that helps make the code readable. And that it's still optional, and that it doesn't affect the runtime semantics of the code, is a pleasant surprise for Spark and his unruly spaghetti twisted code followers.

However, this isn't by any means, Spark's favourite part of the story.

Before I finish this section I'ld like to say a few words on where I suspect Dart's roots began. Strongtalk was a computer language with optional static typing support that was released by Sun in 1997, that's almost twenty years ago, yet there's a strong resemblance between itself and Dart. Stongtalk is also linked to the V8 JavaScript engine and JavaScript. All this to say that Dart's not so much a new language and one returning for a second go at bat!

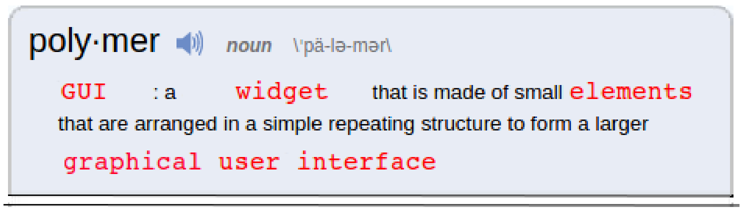
Polymer Comes Into the Picture

IN THE LAND OF BABYLON, there's many languages, and even more frameworks.

Programmers, at least some of us, like to write the least code possible for the biggest buck. I'ld even hazard a guess that few of us consider HTML a language or using it, programming. (Spark, our editor, is barking. Read my comments at the end of this chapter to understand my objections.) Seems to me that most of the frameworks exits to work around the problems of tying together HTML, The DOM, and God know what else, by patching in code. Usually JavaScript. Hence JavaScript's identity as a scripting language.

Cut to the chase: Google gets feed-up with this mess, after years of writing Apps for the web and comes up with < AngularJS >, data-binding, a MVC framework, and a host of other goodies. How can anyone improve on this?

Simple. Forget about all these frameworks and its globs of glue-code and return to the web's beginnings where the HTML < element > was a first class citizen. So when one of the project leaders describe the concept as awesome, he's not kidding. It's the elegance and brilliance of simplicity. In polymer everything is an element, bigger more complex elements are composed of one or more smaller, simpler, element(s). We can even draw parallels in the language of chemistry to software design, particularly to a graphical user interface.


poly·mer noun \ˈpä-lə-mər\

GUI : a widget that is made of small elements that are arranged in a simple repeating structure to form a larger graphical user interface

Polymer GUI

Let's clarify that it's not about HTML and JavaScript scripting in web pages that Polymer targets. It's about building full fledged software applications in a web/browser environment. For example Google's Gmail; although here it's an online application which requires a network connection. Of course the objective for us is to be able to eliminate this restriction and be able to build web applications that will rival desktop applications.

Polymer brings object oriented programming principles to the table.

```
<link rel="import"
      href="/components/polymer/polymer.html">

<polymer-element name="proto-element">
  <template>
    <span>I'm <b>proto-element</b>. Check out my prototype.</span>
  </template>
  <script>
    Polymer('proto-element', {
      ready: function() {
        //...
      }
    });
  </script>
</polymer-element>
```

Polymer Create Element

```
<!DOCTYPE html>
<html>
  <head>
    <script src="platform.js"></script>
    <link rel="import" href="proto-element.html">
  </head>
  <body>
    <proto-element></proto-element>
  </body>
</html>
```

Polymer Apply Element

As an intro to Spark I don't want to get into the details about Polymer other than to say that the line

```
<script src='platform.js'></script>
```

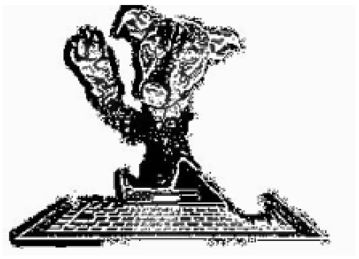
tells us that it's still part of the JavaScript world. Polymer and JavaScript is at the core of this element building framework, if I can stretch the idea of what's a framework. What's interesting and relevant is that the Spark IDE has integrated a Dart language version of Polymer called Polymer.dart, and gives it the description: build structured, encapsulated, client-side web apps with Dart and web components.

A Dart port of Polymer, already?

Yes. Spark brings a large open-source Dart and Polymer project to the table. For developers curious about the Dart programming language and Polymer library, Spark gives them both a young but capable editor and a learning environment all in one place.

Yes. Out of all the JavaScript frameworks to choose Google has picked Polymer in order to develop their own IDE, Spark. Even more interesting is the idea that by setting up and hacking away on this IDE, I get a first class test case of building a Chrome App. Everything in one place as a working example and even an example that I can modify into a working program of my own design.

Unfortunately there's another reference to JavaScript in the Spark IDE; therefore knowing your prototype-based model is helpful in understanding how a Chrome App starts its life.



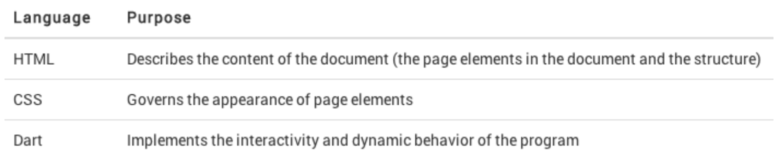
Spark

Spark's objections addressed:

From the official Dart Docs, it reads…

About the HTML, CSS and Dart triumvirate#

Typically three files—an HTML file, a Dart file, and a CSS file—together implement a Dart web application. Each is written in a different language and each is responsible for a different aspect of the program:



Language	Purpose
HTML	Describes the content of the document (the page elements in the document and the structure)
CSS	Governs the appearance of page elements
Dart	Implements the interactivity and dynamic behavior of the program

Languages

HTML is a language for describing web pages. Using tags, HTML sets up the initial page structure, puts elements on the page, and embeds any scripts for page interactivity. HTML sets up the initial document tree and specifies element types, classes, and IDs, which allow HTML, CSS, and Dart programs to refer to the same elements.

CSS, which stands for Cascading Style Sheets, describes the appearance of the elements within a document. CSS controls many aspects of formatting: type face, font size, color, background color, borders, margins, and alignment, to name a few.

Dart code is embedded into an HTML file as a script. A Dart program can respond to events such as mouse clicks, manipulate the elements on a web page dynamically, and can save information.

So where do I get the idea that HTML is not a language?

In computer science we lean about Imperative and Declarative styles of programming. In the beginning Imperative style machine coding was all that we had available, a tedious way of making the machine do something useful for us. Skip ahead to Assembly, to C or BASIC, and we begin to see abstractions of the basic computer functions we had at our disposal. For example, Print(something), as we know is a Declarative description of the underlying Imperative coding that's involved in carrying out the task at hand. Step even farther ahead in time and the levels of abstractions between the coding and the computer language itself become greater.

The language of HTML which was originally described as a good example of a Declarative style of programming began its life with a clear understanding of where it fit in the scheme of computer languages.

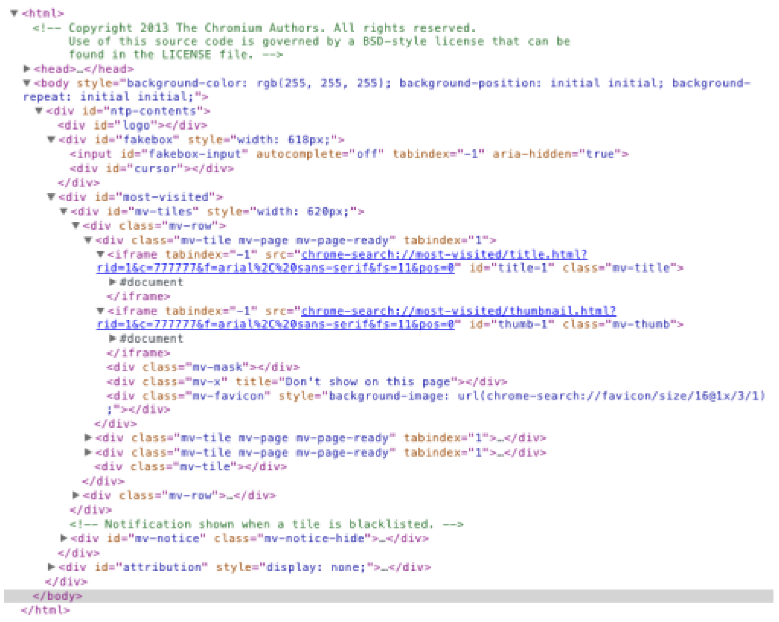
Take the example of an HTML select tag:

```
<select>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

Select

In this Declarative style of HTML coding the understanding is clear by its description: a choice of four options will be presented to the user of which one may be selected. Pretty simple. All the code that implements this functionality is hidden, or encapsulated, from us at this level of HTML coding.

Let's use our tools and look at some of Google's HTML code.



```
▼<html>
   <!-- Copyright 2013 The Chromium Authors. All rights reserved.
        Use of this source code is governed by a BSD-style license that can be
        found in the LICENSE file. -->
  ▶<head>…</head>
  ▼<body style="background-color: rgb(255, 255, 255); background-position: initial initial; background-
  repeat: initial initial;">
    ▼<div id="ntp-contents">
       <div id="logo"></div>
      ▼<div id="fakebox" style="width: 618px;">
         <input id="fakebox-input" autocomplete="off" tabindex="-1" aria-hidden="true">
         <div id="cursor"></div>
       </div>
      ▼<div id="most-visited">
        ▼<div id="mv-tiles" style="width: 620px;">
          ▼<div class="mv-row">
            ▼<div class="mv-tile mv-page mv-page-ready" tabindex="1">
              ▼<iframe tabindex="-1" src="chrome-search://most-visited/title.html?
              rid=1&c=777777&f=arial%2C%20sans-serif&fs=11&pos=0" id="title-1" class="mv-title">
                ▶#document
                </iframe>
              ▼<iframe tabindex="-1" src="chrome-search://most-visited/thumbnail.html?
              rid=1&c=777777&f=arial%2C%20sans-serif&fs=11&pos=0" id="thumb-1" class="mv-thumb">
                ▶#document
                </iframe>
                <div class="mv-mask"></div>
                <div class="mv-x" title="Don't show on this page"></div>
                <div class="mv-favicon" style="background-image: url(chrome-search://favicon/size/16@1x/3/1)
                ;"></div>
              </div>
            ▶<div class="mv-tile mv-page mv-page-ready" tabindex="1">…</div>
            ▶<div class="mv-tile mv-page mv-page-ready" tabindex="1">…</div>
              <div class="mv-tile"></div>
            </div>
          ▶<div class="mv-row">…</div>
          </div>
          <!-- Notification shown when a tile is blacklisted. -->
        ▶<div id="mv-notice" class="mv-notice-hide">…</div>
        </div>
      ▶<div id="attribution" style="display: none;">…</div>
      </div>
    </body>
  </html>
```

Google

I don't know about you but I get the feeling that somewhere the idea of this being Declarative, or that its roots began in a Declarative style of programming, somehow got lost?

That's why I don't think HTML, as it stands today, as computer programming. So Bark all you want, Spark, I'm entitled to my opinion. Though, that's about to change!

Chapter Three: Paradox

Before continuing with Dart and the Polymer Library it’s necessary to understand the JavaScript code which implements the windowless top part of Spark Application.

Chapter Four: ?

Chapter Five: ?

Chapter Six: ?

Chapter Seven: Testing

Chapter Eight: Deployment

Chapter Nine: What’s next?