



Dart

... and why you should care
... but probably don't
... yet

Kasper Lund

GOTO Nights
September, 2014

[#dartlang](#)



Who am I?

Kasper Lund, software engineer at Google
Co-founder of the Dart project

Key projects

V8: High-performance JavaScript engine

Dart: Structured programming for the web





Dart

What is it, really?



TL;DR

Programming language

Integrated development tools

Rich core libraries



TL;DR

Programming **language**

Integrated development **tools**

Rich core **libraries**



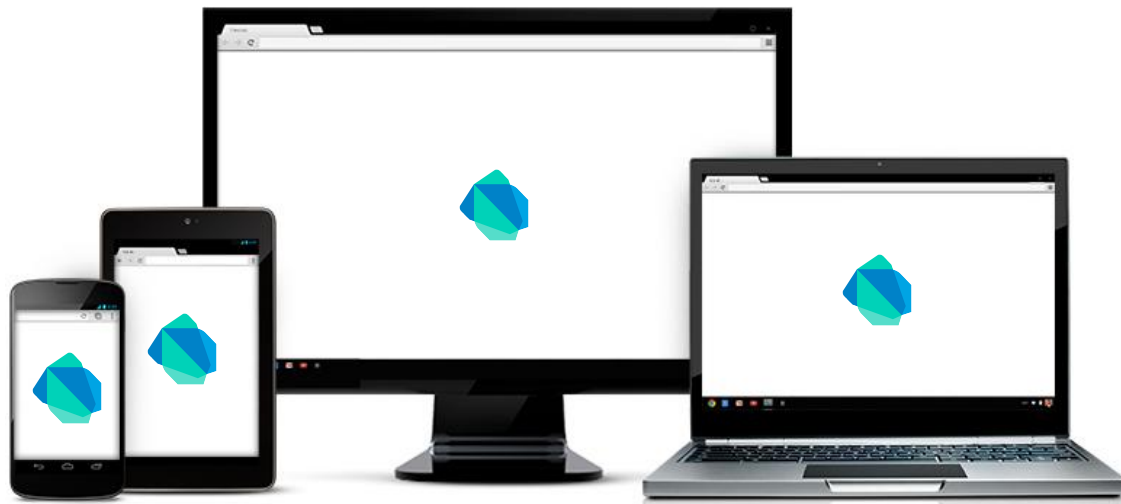


Dart 1.0

Dart is a scalable web app platform



Dart runs everywhere!



Runs on native Dart VM - or translated to JavaScript



Runs on native Dart VM



Language



The Dart language

Unsurprising and object-oriented

Class-based single inheritance

Familiar syntax with lexical scoping

Optional static type annotations



Dart for JavaScript programmers

```
main() {  
  var greeting = "Hello, World";  
  print(greeting);  
}
```

Dart is flexible

Let's change this to appeal a bit more to Java programmers



Dart for Java programmers

```
void main() {  
  String greeting = "Hello, World";  
  print(greeting);  
}
```

What? No classes?



Dart for Java programmers

```
void main() {  
  Person person = new Person("Kasper");  
  print("Hello $person");  
}
```

```
class Person {  
  String name;  
  Person(this.name);  
  toString() => name;  
}
```



Dart for JavaScript programmers

```
main() {  
  var person = new Person("Kasper");  
  print("Hello $person");  
}
```

```
class Person {  
  var name;  
  Person(this.name);  
  toString() => name;  
}
```

Proper lexical scoping

No reason to write `this.name` here!

Fail early and predictably

Typos lead to recognizable compile-time and runtime errors





Standard ECMA-408

1st Edition / June 2014

Dart Programming Language Specification

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001



#dartlang

Tools



The Dart tools

Working with code

Analyzer

Editor

Formatter

Package manager (*pub*)

Executing code

Virtual machine

Dart-to-JavaScript compiler (*dart2js*)

Understanding code

Coverage tracker

Profiler

Debugger



Let's see that in action!

Demonstration of the Dart editor



#dartlang

Toolability

What makes a language toolable?

1. The language must have enough structure to allow efficient static analysis
2. The language must allow developers to understand and trust the analysis results



Libraries



The Dart libraries




Core libraries - dart:core

Classes: bool, int, double, String

```
value.clamp(0, 255).toRadixString(16)
```

```
input.trim().padLeft(8, padding: "0")
```



Named optional
parameter



Core libraries - dart:core

Classes: List, Map, Set

```
Map<int, String> cache = ...;  
cache.keys  
    .where((key) => key.isEven)  
    .map((key) => cache[key])  
    .forEach(print);
```

All the intermediate
collections are lazy and
of type Iterable



Core libraries - dart:async

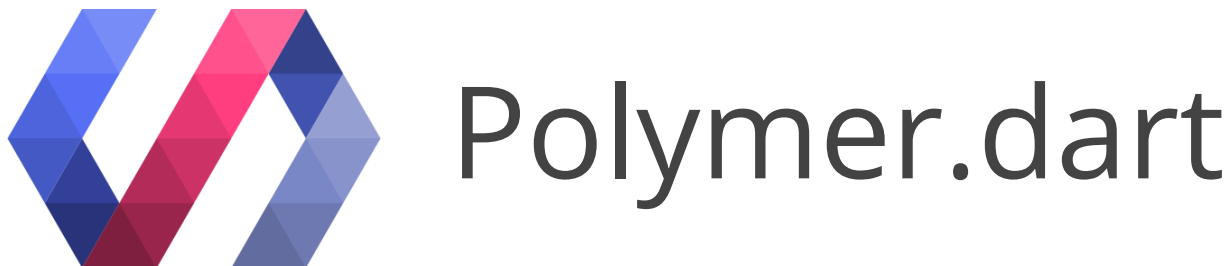
Classes: Future, Stream

```
HttpRequest.getString("localhost:8080")  
  .then(print);
```

```
input.onKeyUp  
  .map((event) => event.target.value)  
  .distinct()  
  .forEach(print);
```



Higher level libraries



pub.dartlang.org



© thejetsetter.co.uk

#dartlang



Dart everywhere!

Demo of client- and server-side code



Language

(part deux)



New language features

Enumerations

Generators and async functions

Deferred loading



New language features

~~Enumerations enum~~

Generators and async functions

Deferred loading



The four types of functions

	one	many
sync	T	Iterable<T>
async	Future<T>	Stream<T>



Repeatedly waiting

```
final context = querySelector("canvas").context2D;  
bool running = true; // Set to false to stop animation.
```

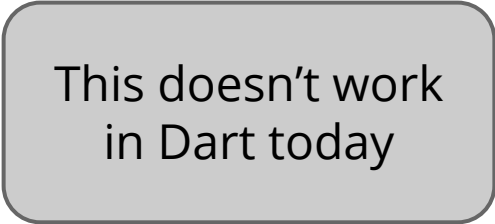
```
main() {  
  void tick(time) {  
    context.clearRect(0, 0, 500, 500);  
    context.fillRect(time % 450, 20, 50, 50);  
    if (running) window.animationFrame.then(tick);  
  }  
  window.animationFrame.then(tick);  
}
```



Repeatedly awaiting!

```
final context = querySelector("canvas").context2D;  
bool running = true; // Set to false to stop animation.
```

```
main() async {  
  while (running) {  
    var time = await window.animationFrame;  
    context.clearRect(0, 0, 500, 500);  
    context.fillRect(time % 450, 20, 50, 50);  
  }  
}
```



This doesn't work
in Dart today



Our libraries are ready!

Properties

`Future<num> get animationFrame`

Returns a Future that completes just before the window is about to repaint so the user can draw an animation frame.

If you need to later cancel this animation, use [requestAnimationFrame](#) instead.

The [Future](#) completes to a timestamp that represents a floating point value of the number of milliseconds that have elapsed since the page started to load (which is also the timestamp at this call to `animationFrame`).

Note: The code that runs when the future completes should call [animationFrame](#) again for the animation to continue.



The four types of functions

	one	many
sync	T	Iterable<T>
async	Future<T>	Stream<T>



The four types of functions

	one	many
sync	T	Iterable<T>
async	(..) async { ... }	Stream<T>



The four types of functions

		yield
	(..) { ... }	(..) sync* { ... }
await	(..) async { ... }	(..) async* { ... }

```
Iterable<int> range(int n) sync* {  
  for (int i = 0; i < n; i++) {  
    yield i;  
  }  
}
```



Deferred loading



Deferred loading

```
import "analytics.dart" deferred as analytics;
```

```
main() {  
  startAnalytics(); // Doesn't block.  
  startApplication();  
}
```

```
startAnalytics() async {  
  await analytics.loadLibrary();  
  analytics.enableControls();  
}
```



Productivity



The Dart productivity

Expr

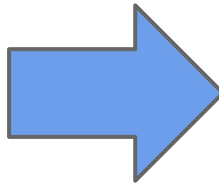
C

Ric

... but is that really the **entire** story?




Unsurprising semantics



Constructors are just functions


```
function Point(x, y) {  
  this.x = x;  
  this.y = y;  
}
```

Forgot to write **new**



```
var point = Point(2, 3);  
assert(point == undefined);  
assert(x == 2 && y == 3);
```

So we get two new
global variables!



JS



Accessing non-existing properties

```
var request = new XMLHttpRequest();
```

```
...
```

```
request.onreadystatechange = function() {
```

```
  if (request.readyState == 4) {
```

```
    console.log('Request done: ' + request.responseText);
```

```
  }
```

```
};
```

Did you mean
readyState?



Subtle details

If you find yourself always **worrying** about subtle **details** in your code ...

- ... you are distracted

- ... you are less productive

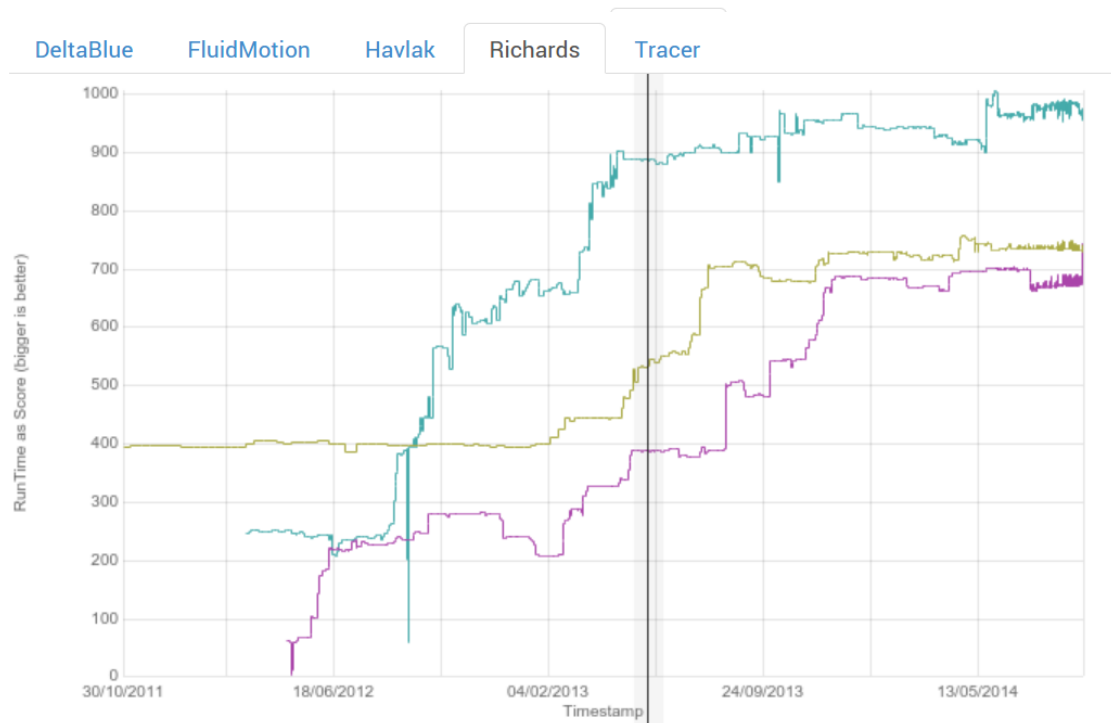
- ... you are ready to try Dart!



Performance



Benchmarks



.dart
.dart.js
.js

#dartlang



Try that with a larger code base...

Code base

Feature

Implement

er, etc.

code

```
Dart (1.6.0): dart2js.dart
```

```
User time (seconds): 17.85
```

```
System time (seconds): 0.95
```

```
Percent of CPU this job got: 102%
```

```
Memory usage (MB): 153
```

```
V8 (3.26.31.10): dart2js.dart.js
```

```
User time (seconds): 49.25
```

```
System time (seconds): 1.79
```

```
Percent of CPU this job got: 120%
```

```
Memory usage (MB): 484
```



Conclusions



Dart is ...

unsurprising

familiar

... and ready to be used today!

object-oriented



Your new platform for the web?

Dart is a stable platform you can use today

It is easy to get started - just visit dartlang.org



Thank you!



#dartlang