# Using TypeScript to develop large JavaScript applications – Experiences from the trenches

Dirk Bäumer

# Introduction to TypeScript

Application scale JavaScript development is hard.

TypeScript is a typed superset of JavaScript
that compiles to plain JavaScript.

Any browser. Any host. Any OS.

Open Source.

# TypeScript

## Starts with JavaScript

All JavaScript code is TypeScript code, simply copy and paste
All JavaScript libraries work with TypeScript

## Optional Static Types, Classes, Modules

Enable scalable application development and excellent tooling
Zero cost: Static types completely disappear at run-time

## Alligned with ECMAScript 6

Classes, Modules, Arrow Functions, Rest Arguments and Default Arguments are aligned with what is currently proposed in ECMAScript 6

## Ends with JavaScript

Compiles to idiomatic JavaScript
Runs in any browser or host, on any OS

# Demo

Getting started with TypeScript

# Writing better JavaScript

## Formalization of common JavaScript patterns

Function overloads, rest and default arguments

## Type inference and structural typing

In practice very few type annotations are necessary

## Works with existing JavaScript libraries

Declaration files can be written and maintained separately

## Not “provably type safe”

Types reflect intent but do not provide guarantees

# Demo

TypeScript Classes and Modules

# Classes and Modules

## Scalable application structuring

Classes, Modules, and Interfaces enable clear contracts between components

## Supports popular module systems

CommonJS and AMD modules in any ECMAScript 3 compatible environment

# What's Included?

## Compiler

Open Source, written in TypeScript

## Tooling

Visual Studio language service, browser hosted playground

## Libraries

Full static typing of DOM, jQuery, node.js, …
See also https://github.com/borisyankov/DefinitelyTyped)

## And More

Lots of samples and formal Language Specification

# Converting form JavaScript to TypeScript

# Growing a JS Code base

"JavaScript code 'rots' over time" --Ben

"Writing JavaScript code in a large project is like carving code in stone" --Alex

"It is easy to write a large JS program it is difficult to maintain it" --Anders

# Surviving JS Growth

## Up to 50k LOC

free style  (with unit tests)

## 50k LOC

use class and module pattern → WinJS classes, name spaces

## > 100k LOC

use explicit module dependencies → AMD or CommonJS

## > 150k LOC

use typed interfaces and add type annotations → TypeScript

# From 30% to 60%

## Define .d files for existing JS code

New code implemented in TypeScript

Used a tool to convert WinJS.define calls to ambient classes

But .d files are hard to maintain as the JS code evolves

## Converted existing code

Bottom-up: start with files that has no dependencies

Type the API surface

- No type inference for API
- Introduce interfaces for object literals and object bags/maps

# Experiences

"As I did conversions, I began typing various object literals I was passing around as interfaces. Soon enough, I realized how inconsistent I was, the same data was flowing around in at least 3 different formats. This is because of the easiness through which you can create literals in JavaScript …. Need some placeholder for data?... Just create a new literal object." --Alex

"In JavaScript, you are really at the mercy of your ability to spell. "

```
function deleteRange(range) {
    delete this.markers[range.statMarkerId]; //start
```

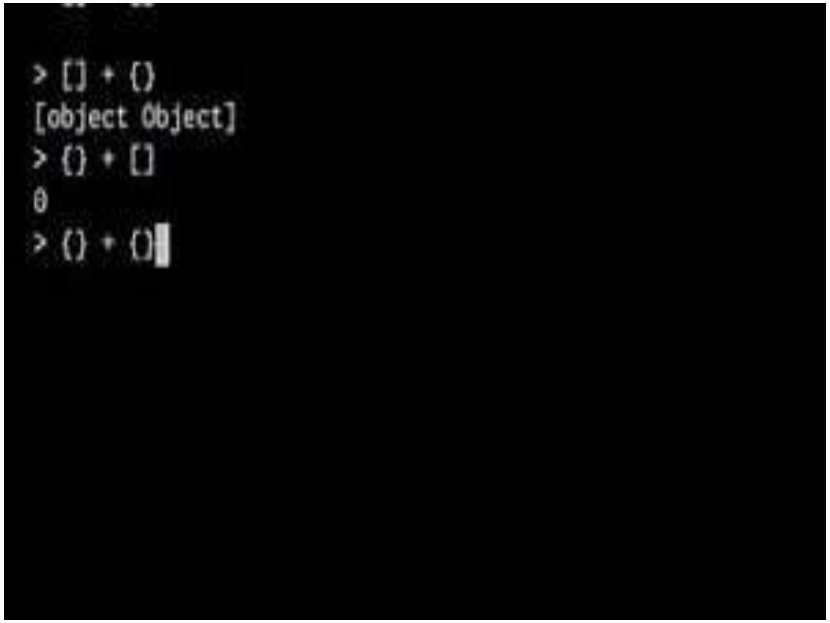
--Alex

"I would really like to benefit from type checking everywhere in the code. I went on to hover over all variables to see if their type was inferred and I helped the compiler out with type annotations only where it failed. --Ben

# Features we love most beside Types, Classes and Modules

| | |
|---|---|
| Arrow functions | Interface with optional properties |
| Typing of Maps | Function overloads |

And some fun
https://www.destroyallsoftware.com/talks/wat



# … and how TypeScript helps with this

```
1 var result = [] + [];
2
3 result = [] + {};
4
5 result = {} + [];
6
7 result = {} + {};
8
9 result = "abc" - 1;
```

```
Operator '-' cannot be applied to types 'string' and 'number'
any
```

Application scale JavaScript development is hard.

TypeScript makes it easier.

http://typescriptlang.org