

Having F#un with JavaScript (FunScript)

Talk by Ramón Soto Mathiesen
@ Open Source Days 2013

Contents

- About me
- What is FunScript ...
- ... and why we should use it
- Live demo: Xamarin Studio and Emacs
- Summary

About me

- MSc. Computer Science from DIKU
- CRM Architect
 - ▶ ER-modeling, WSDL, OData (REST API)
- C++/C#/JavaScript
- Hopefully very soon, mostly F#

What is FunScript ...

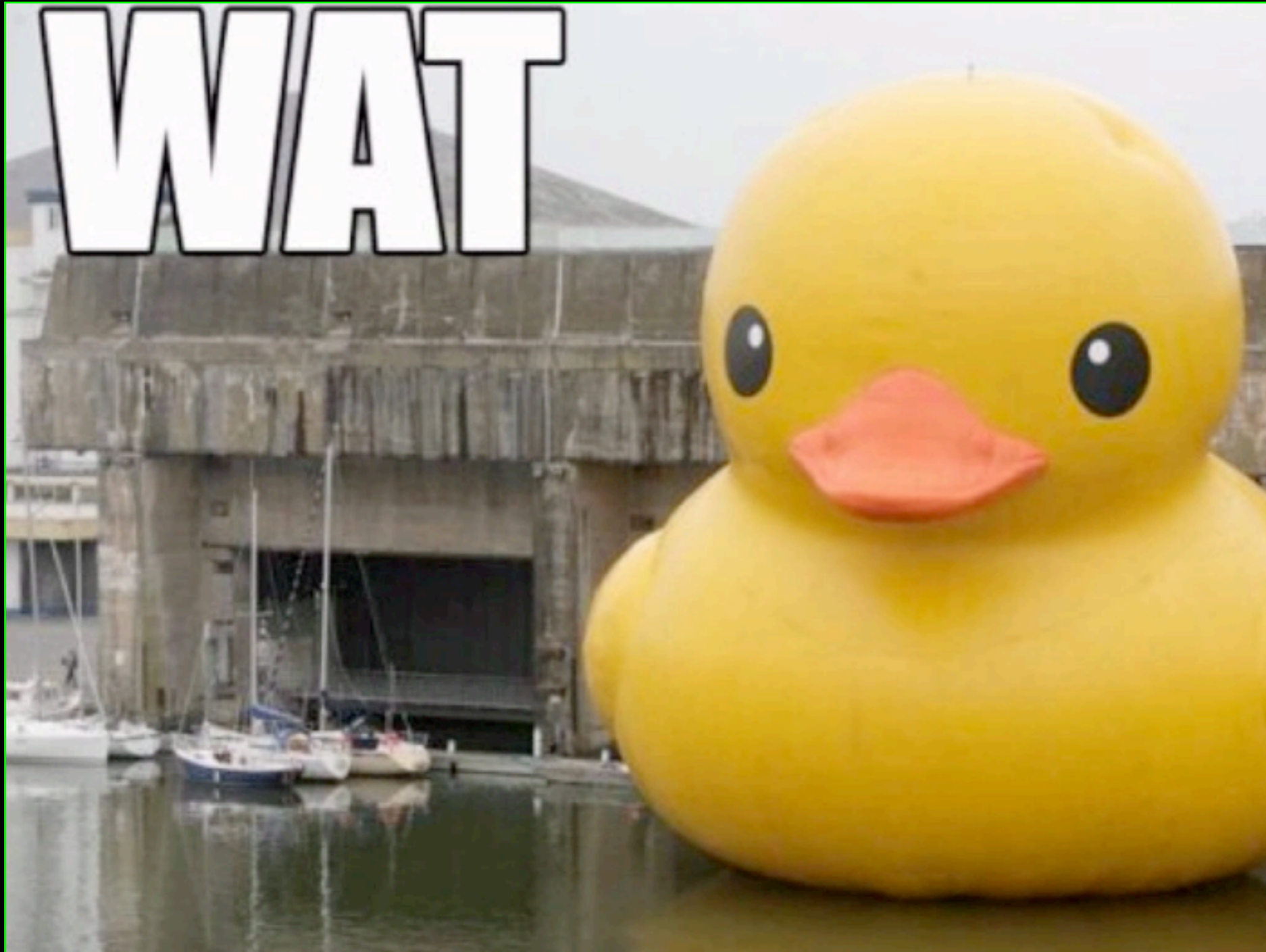
```
// F# Code -> JavaScript Code  
Compiler.Compile: Expr -> string
```

- Mono .NET (Mac, Linux and Windows)
- Created by Zach Bray
 - BSc. in CS from Cambridge
 - Software Developer at Trayport UK
- Other contributors: Tomas Petricek, Phillip Trelford, James Freiwirth, Robert Pickering and Steffen Forkmann (and soon me)

... and why use it

WAT - A lightning talk by Gary
Bernhardt @ CodeMash 2012

... and why use it



... and why use it

- JS is a dynamic language
- Strange behavior in the type-system (WAT)
- JS is the only programming language present on all devices OS browsers (iOS, Android, Windows) but now also as desktop apps (HTML5) or server-side software (node.js)
- Errors are caught on compile-time and not run-time
- Possibility to write in a strongly-typed functional language (that is F#un right?)

... and why use it

small quiz - what is the output?

```
foo = "global"

function bar(){
  console.log(foo);
  var foo = "local";
  console.log(foo);
}

bar();

console.log(foo);
```


... and why use it

small quiz - output

```
undefined  
local  
global
```

... and why use it

small quiz - explanation

```
foo = "global"

function bar(){
  var foo; // WAT
  console.log(foo);
  foo = "local";
  console.log(foo);
}

bar();

console.log(foo);
```

What does it support?

- Most F# code
- A little mscorlib
- 400+ bootstrapped tests
 - ▶ FSharp.PowerPack
 - ▶ Jint - Javascript Interpreter for .NET

Most F# code

Primitives

- Strings
- Numbers (beware!)
- Booleans

Most F# code

Flow

```
let y =  
    if x then "foo"  
    else "bar"  
y
```

```
var _temp1;  
if (x)  
{  
    _temp1 = "foo";  
}  
else  
{  
    _temp1 = "bar";  
};  
var y = _temp1;  
return y;
```

```
let xs = [1; 2; 3]  
match xs with  
| x::xs -> x  
| _ -> failwith "never"
```

```
var xs = List_CreateCons(1.000000,  
List_CreateCons(2.000000,  
List_CreateCons(3.000000,  
List_Empty())));  
if ((xs.Tag == "Cons"))  
{  
    var _xs = List_Tail(xs);  
    var x = List_Head(xs);  
    return x;  
}  
else  
{  
    throw ("never");  
}
```

Most F# code

Functions

```
let isOdd x = x % 2 <> 0
isOdd 2
```

```
var isOdd = (function (x)
{
    return ((x % 2.000000).CompareTo(0.000000) != 0.000000);
});
return isOdd(2.000000);
```

```
(fun x -> x % 2 = 0)(2)
```

```
return (function (x)
{
    return ((x % 2.000000).CompareTo(0.000000) = 0.000000);
})(2.000000);
```

Most F# code

Records

```
type Person =  
    { Name: string; Age: int }  
let bob =  
    { Name = "Bob"; Age = 25 }
```

```
var i_Person__ctor;  
i_Person__ctor = (function (Name, Age)  
    {  
        this.Name = Name;  
        this.Age = Age;  
    });  
var bob = (new i_Person__ctor("Bob", 25.000000));
```

```
let now = { Name = "Bob"; Age = 25 }  
let soon = { now with Age = 26 }
```

```
var now = (new i_Person__ctor("Bob", 25.000000));  
var _temp1;  
var Age = 26.000000;  
_temp1 = (new i_Person__ctor(now.Name, Age));  
var soon = _temp1;
```

...but also discriminated unions, classes and modules

Most F# code

Operators

```
let xs = [10 .. 20]
```

```
let xs = [10 .. 2 .. 20]
```

```
let incr x = x + 1  
let x = 10  
x |> incr
```

```
let incr x = x + 1.  
let divBy2 x = x / 2.  
(incr << divBy2) 10.
```

```
var xs = Seq.ToList(Range_oneStep(10.000000, 20.000000));
```

```
var xs = Seq.ToList(Range_customStep(10.000000, 2.000000, 20.000000));
```

```
var incr = (function (x)  
{  
    return (x + 1.000000);  
});  
var x = 10.000000;  
return incr(x)
```

```
var incr = (function (x)  
{  
    return (x + 1.000000);  
});  
var divBy2 = (function (x)  
{  
    return (x / 2.000000);  
});  
return (function (x)  
{  
    return incr(divBy2(x));  
})(10.000000);
```


Most F# code

Computation expressions

```
async { return () }  
|> Async.StartImmediate
```

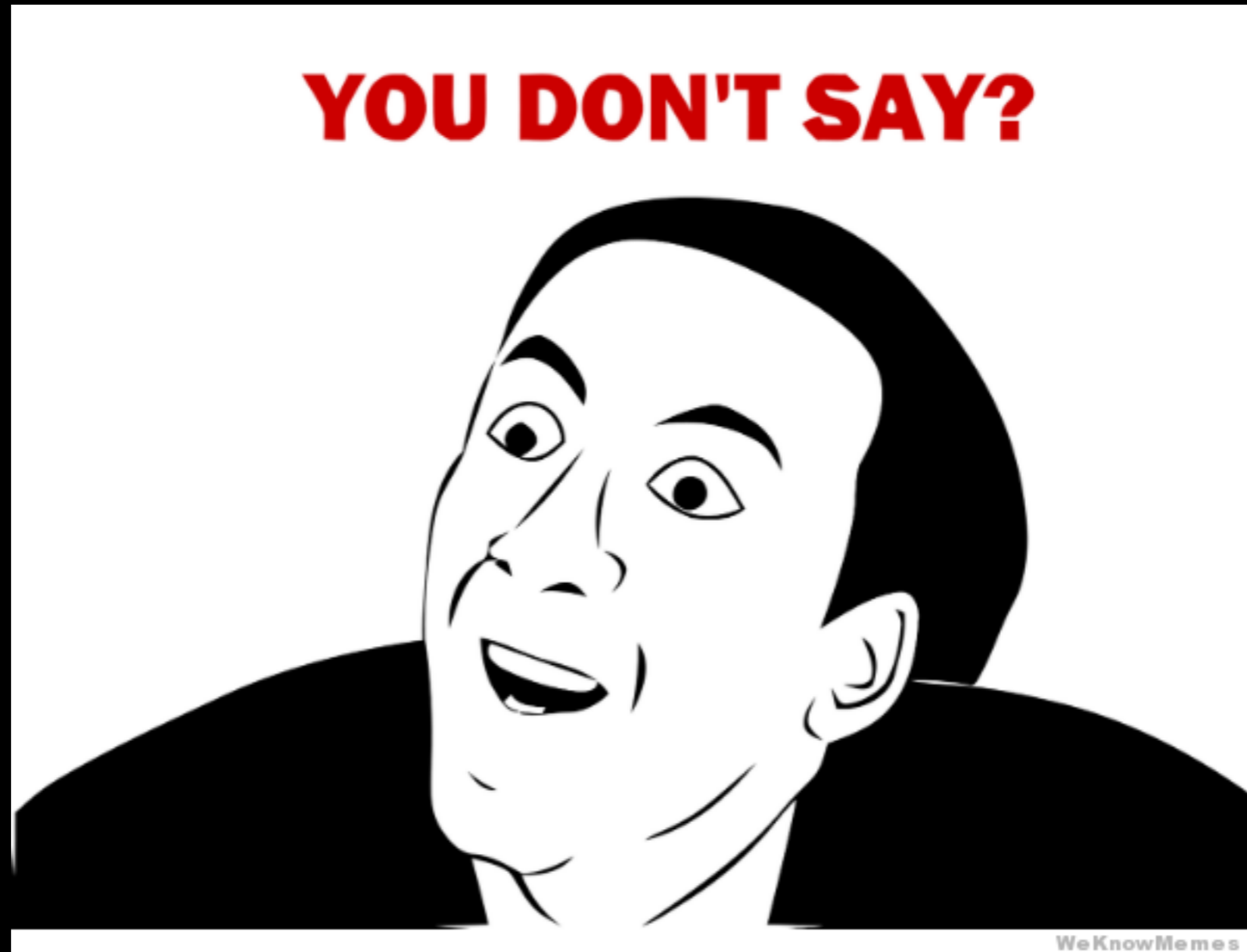
```
return (function (arg00)  
{  
  return Async_StartImmediate(arg00, {Tag: "None"});  
})((function (builder_)  
{  
  return builder_.Delay((function (unitVar)  
  {  
    var _temp3;  
    return builder_.Return(_temp3);  
  }));  
}))(Async_get_async()));
```

Most F# code

Data structures

- *Array*
- List
- Seq
- Map
- Set
- Option

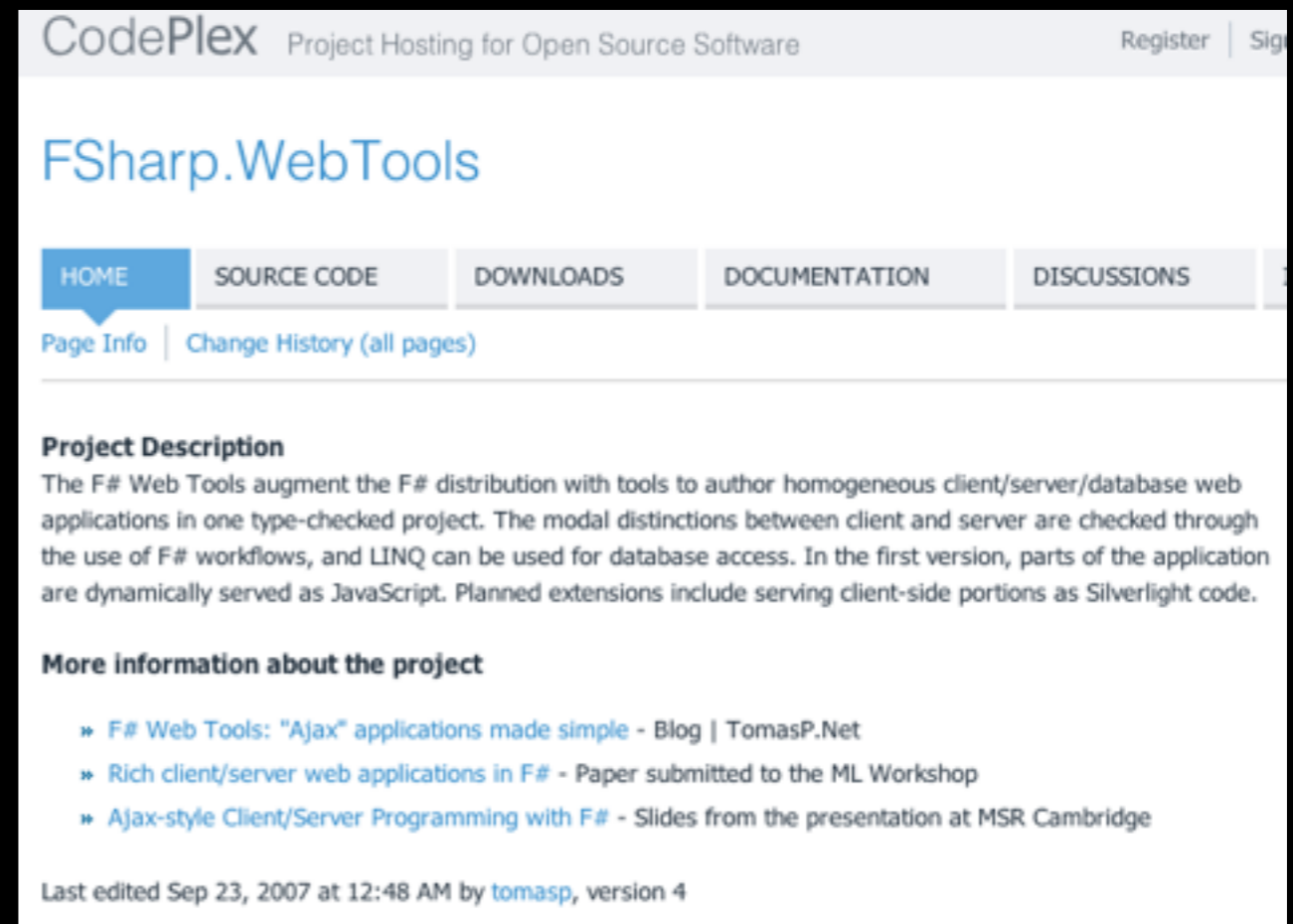
So strongly-type to JS ...
... nothing new under the Sun



So strongly-type to JS ...

... nothing new under the Sun

- F# frameworks
 - ▶ FSWebTools
 - ▶ WebSharper
 - ▶ Pit
 - ▶ JSIL



The screenshot shows the CodePlex project page for FSharp.WebTools. The page header includes the CodePlex logo and navigation links for Register and Sign In. The main title is "FSharp.WebTools". Below the title is a navigation menu with tabs for HOME, SOURCE CODE, DOWNLOADS, DOCUMENTATION, and DISCUSSIONS. The HOME tab is selected. Below the navigation menu are links for Page Info and Change History (all pages). The main content area contains a "Project Description" section, which states: "The F# Web Tools augment the F# distribution with tools to author homogeneous client/server/database web applications in one type-checked project. The modal distinctions between client and server are checked through the use of F# workflows, and LINQ can be used for database access. In the first version, parts of the application are dynamically served as JavaScript. Planned extensions include serving client-side portions as Silverlight code." Below the description is a "More information about the project" section with three links: "F# Web Tools: 'Ajax' applications made simple - Blog | TomasP.Net", "Rich client/server web applications in F# - Paper submitted to the ML Workshop", and "Ajax-style Client/Server Programming with F# - Slides from the presentation at MSR Cambridge". At the bottom of the page, it says "Last edited Sep 23, 2007 at 12:48 AM by tomasp, version 4".

So strongly-type to JS ...

... other Frameworks



Elm

So strongly-type to JS ...

Dynamically typed

- Good at interoperability
(use JavaScript libraries and access data)
- But if its too close to JavaScript...

```
> '2' * 3
6
> '2' + 1
21
```

```
> for(var x in [3,4,5])
  console.log(x);
0
1
2
```

```
> var x = 10;
  {
    var x = 11;
  }
  console.log(x);
11
```

```
> var x = NaN;
  var y = 1;
  undefined
> if(x) console.log(x);
  x == false;
  false
> if(y) console.log(y);
  y == false;
  1
< false
> x == true;
  false
> y == true;
  true
```

```
> var f = function() {
  return {
    foo: "bar"
  };
};
var g = function() {
  return
  {
    foo: "bar"
  };
};
undefined
> f();
▶ Object
> g();
undefined
```

So strongly-type to JS ...

Dynamically typed



So strongly-type to JS ...

Statically typed

- Foreign function interface (FFI)
- Map every function that you want to use
- Tedious and error prone
(you might be better off going dynamic)
- Cannot easily access any of the existing JavaScript infrastructure
- If you have to use Foreign function interface (FFI) you are a lonely island

... so why should we “really” use it

- FunScripts focus is on Extensibility
 - ▶ Impossible to port the whole framework
 - ▶ Instead provide tools to make the parts you need.

... so why should we “really” use it

- Bypass FFI with type providers
- F# is the only language that supports this workflow at the moment!

```
open Microsoft.FSharp.Linq
open Microsoft.FSharp.Data.TypeProviders

type Netflix = ODataService<"http://odata.netflix.com/Catalog/">
let data = Netflix.GetDataContext()

let topMovies = query {
  for movie in data.
}
    Credentials
    DataContext
    Genres
    Languages
    People
    TitleAudioFormats
    TitleAwards
    TitleScreenFormats
    Titles
val data : Netflix.ServiceTypes.SimpleDataContextTypes.NetflixCatalog
```

... so why should we “really” use it

- The TypeScript library creates a bunch of types and tells the compiler how to turn them into JavaScript.

```
open FunJS
open FunJS.TypeScript

// See https://github.com/borisyankov/DefinitelyTyped for more
type ts = FunJS.TypeScript.Api<""
    ../../Typings/jquery.d.ts
    ../../Typings/google.maps.d.ts
    ../../Typings/lib.d.ts"" >

type gmaps = ts.google.maps
```

gmaps.M

- ImageMapType
- ImageMapTypeOptions
- KmlLayerMetadata
- KmlMouseEvent
- MVCArray**
- MVCObject
- Map
- MapCanvasProjection
- MapOptions

```
type MVCArray =
  new : unit -> MVCArray
  member clear : unit -> Unit
  member forEach : callback: obj * float -> Unit -> Unit
  member getArray : unit -> obj []
  member getAt : i: float -> obj
```

Live demo:

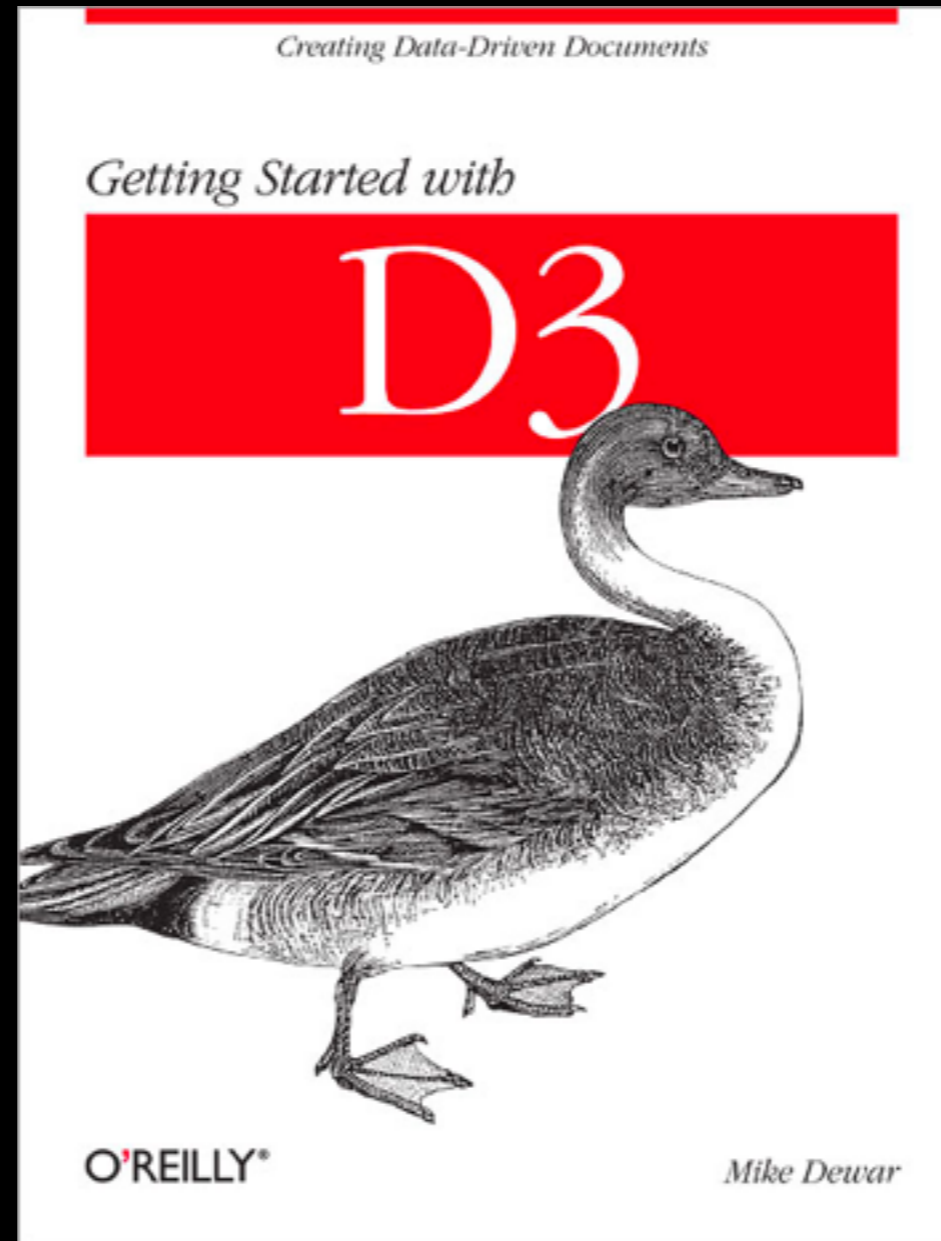
Xamarin Studio and Emacs

```
File Edit Options Buffers Tools F# Help
1  [<ReflectedDefinition>]
2
3  module Program
4
5  open FunScript
6  open FunScript.TypeScript
7
8  // -----
9  // Initializataion
10
11 type j    = FunScript.TypeScript.Api< @"/D3/lib/ts/jquery.d.ts" >
12 type jui = FunScript.TypeScript.Api< @"/D3/lib/ts/jqueryui.d.ts" >
13 type h    = FunScript.TypeScript.Api< @"/D3/lib/ts/highcharts.d.ts" >
14 type d3   = FunScript.TypeScript.Api< @"/D3/lib/ts/d3.d.ts" >
15
16 let jQuery (command:string) = j.jQuery.Invoke(command)
17 let (?) jq name = jQuery("#" + name)
18
19 let fooDivTag = jQuery?fooDivTag.html()
20 let barDivTag = jQuery?barDivTag.html2()
21
22 // -----
```

```
<<EOF>>
errors
INFO: Background parsing started
<<EOF>>
DATA: errors
[19:33-19:38] ERROR The field, constructor or member 'html2' is not defined
<<EOF>>
```

Live demo:

Xamarin Studio and Emacs



Example code: <http://examples.oreilly.com/0636920025429/>

FunScript:

Lessons learned

- FunScript.TypeScript.Api bug
(needs to be called with “../current path”)
- D3 TypeScript declaration file not fully implemented
 - ▶ Not possible to make the books basic examples
- FunScript don't support TypeScript 100%:
 - ▶ `x: { (foo: string) : string; (bar:number) : number; };`
 - ▶ `interface Foo extends Bar`
- Difficult to debug. Everything *failwith "never"* is not helpful
- It's not a *showstopper*, but there room for improvements
 - ▶ I will become a contributor :) what about you?

Summary

- It supports almost all of F#.
- It is extensible (re-route any method call), which allows "Information Rich Programming".
- F# is the **only** statically typed language that can do this at the moment. Everything else (statically typed) relies on FFI definitions, which has a poor maintenance story and is not scalable.
- FunScript will be looking at consuming JavaScript through a type provider to the node package manager soon.
- How to contribute?
 - ▶ <https://github.com/ZachBray/FunScript>

Questions?