TypeScript Introduction

TypeScript

JavaScript on Steroids

Rainer Stropek

software architects gmbh

Mail	rainer@timecockpit.com
Web	http://www.timecockpit.com
Twitter	@rstropek

time cockpit
Saves the day.

Why TypeScript?

- JavaScript is great because of its reach

  JavaScript is everywhere

- JavaScript is great because of available libraries

  For server and client

- JavaScript (sometimes) sucks because of missing types

  Limited editor support (IntelliSense)

  Runtime errors instead of compile-time errors

- Our wish: Productivity of robustness of C# with reach of JavaScript

What is TypeScript?

- Valid JavaScript is valid TypeScript
  - TypeScript defines add-ons to JavaScript (primarily type information)
  - Existing JavaScript code works perfectly with TypeScript
- TypeScript compiles into JavaScript
  - Compile-time error checking base on type information
  - Use it on servers (with node.js), in the browser, in Windows Store apps, etc.
  - Generated code follows usual JavaScript patterns (e.g. pseudo-classes)
- Microsoft provides great tool support
  - E.g. IntelliSense in VS2012

Typing Basics

Any

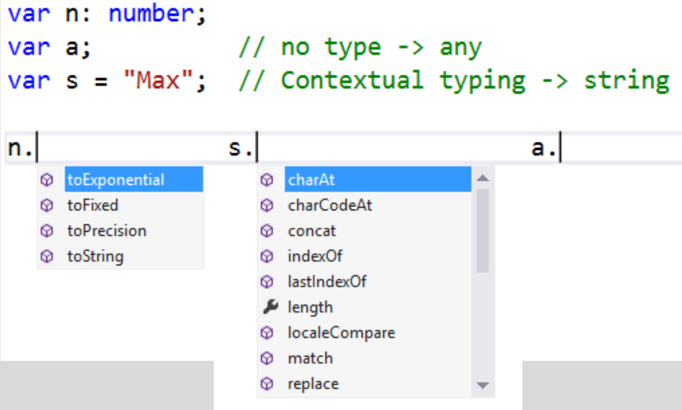
Primitive Types

- *Number*
- *Boolean*
- *String*

Object Types

- Classes, Modules, Interfaces, …

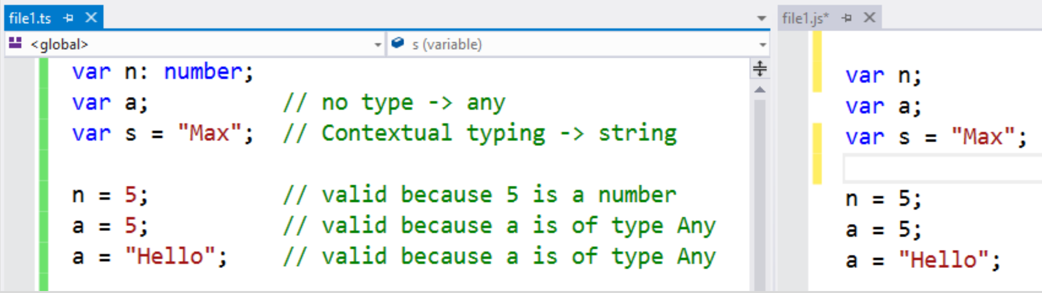
```
var n: number;
var a;                // no type -> Any
var s = "Max";        // Contextual typing -> string

n = 5;                // valid because 5 is a number
a = 5;                // valid because a is of type Any
a = "Hello";          // valid because a is of type Any
n = "Hello";          // compile time error because
                      // "Hello" is not a number
```



```
var n: number;
var a;          // no type -> any
var s = "Max";  // Contextual typing -> string

n.          s.                 a.
```

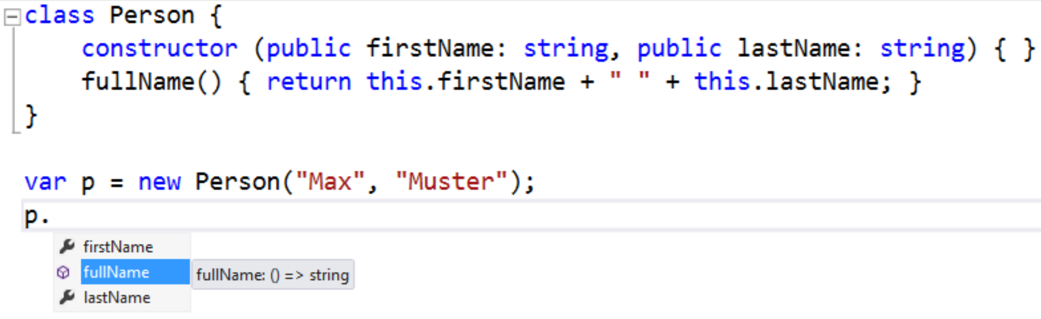
VS2012 IntelliSense based on types

TypeScript Introduction



```
var n: number;
var a;              // no type -> any
var s = "Max";  // Contextual typing -> string

n = 5;              // valid because 5 is a number
a = 5;              // valid because a is of type Any
a = "Hello";     // valid because a is of type Any
```

```
var n;
var a;
var s = "Max";

n = 5;
a = 5;
a = "Hello";
```



```
class Person {
    constructor (public firstName: string, public lastName: string) { }
    fullName() { return this.firstName + " " + this.lastName; }
}

var p = new Person("Max", "Muster");
p.
```

What happens with types in JavaScript?

No performance impact ☺

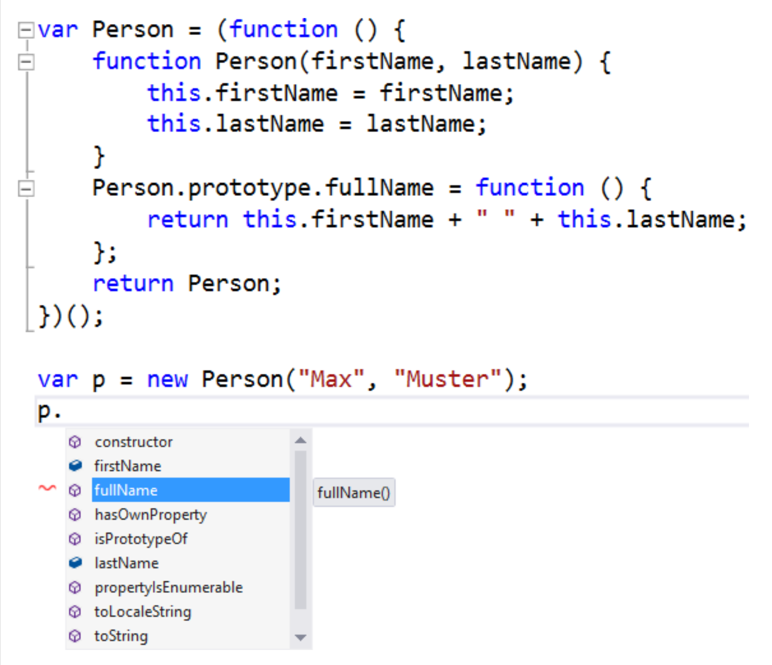
Typing Basics

Types are used during editing and compiling

- No type information in resulting JavaScript code

Contextual Typing

- Determine result type from expressions automatically



```
var Person = (function () {
    function Person(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    Person.prototype.fullName = function () {
        return this.firstName + " " + this.lastName;
    };
    return Person;
})();

var p = new Person("Max", "Muster");
p.
```

What happens with classes in JavaScript?

Results in the usual JavaScript pseudo-class pattern

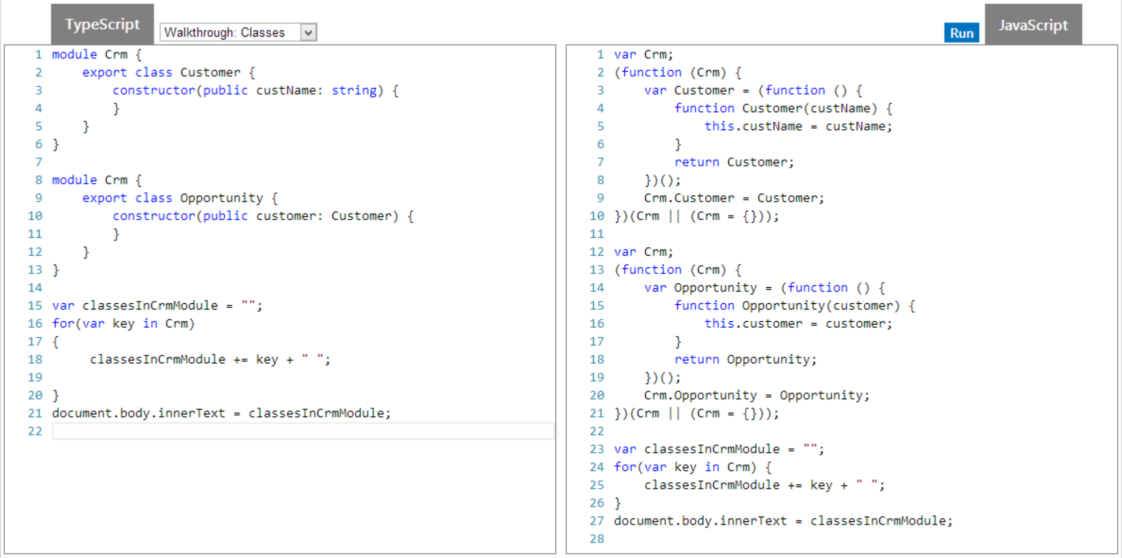
Typing Basics

TypeScript classes become JavaScript pseudo-classes

http://javascript.info/tutorial/pseudo-classical-pattern

TypeScript Introduction

Typing Basics



```typescript
module Crm {
    export class Customer {
        constructor(public custName: string) {
        }
    }
}

module Crm {
    export class Opportunity {
        constructor(public customer: Customer) {
        }
    }
}

var classesInCrmModule = "";
for(var key in Crm)
{
     classesInCrmModule += key + " ";

}
document.body.innerText = classesInCrmModule;

```

```javascript
var Crm;
(function (Crm) {
    var Customer = (function () {
        function Customer(custName) {
            this.custName = custName;
        }
        return Customer;
    })();
    Crm.Customer = Customer;
})(Crm || (Crm = {}));

var Crm;
(function (Crm) {
    var Opportunity = (function () {
        function Opportunity(customer) {
            this.customer = customer;
        }
        return Opportunity;
    })();
    Crm.Opportunity = Opportunity;
})(Crm || (Crm = {}));

var classesInCrmModule = "";
for(var key in Crm) {
    classesInCrmModule += key + " ";
}
document.body.innerText = classesInCrmModule;

```

How do modules work?

Results in the usual JavaScript module pattern

TypeScript Introduction

```
module CrmModule {
  // Define an interface that specifies
  // what a person must consist of.
  export interface IPerson {
    firstName: string;
    lastName: string;
  }
  …
}
```

Language Overview

Modules

Interfaces

TypeScript Introduction

```
export class Person implements IPerson {
  private isNew: bool;
  public firstName: string;

  constructor(firstName: string, public lastName: string) {
    this.firstName = firstName;
  }

  public toString() { return this.lastName + ", " + this.firstName; }

  public get isValid() {
    return this.isNew ||
      (this.firstName.length > 0 && this.lastName.length > 0);
  }

  public savePerson(repository, completedCallback: (bool) => void) {
    var code = repository.saveViaRestService(this);
    completedCallback(code === 200);
  }
}
```

Language Overview

Classes

Note that Person would not need to specify *implements IPerson* explicitely. Even if the *implements* clause would not be there, *Person* would be compatible with *IPerson* because of structural subtyping.

Constructor

Note the keyword *public* used for parameter *lastName*. It makes *lastName* a public property. *FirstName* is assigned manually.

Function Type Literal

Note the function type literal used for the *completeCallback* parameter. *repository* has no type. Therefore it is of type *Any*.

```
// Create derived classes using the "extends" keyword
export class VipPerson extends Person {
  public toString() {
    return super.toString() + " (VIP)";
  }
}
```

Language Overview

Derived Classes

Note that *VipPerson* does not define a constructor. It gets a constructor with appropriate parameters from its base class automatically.

TypeScript Introduction

```
module CrmModule {
  …

   // Define a nested module inside of CrmModule
  export module Sales {
    export class Opportunity {
      public potentialRevenueEur: number;
      public contacts: IPerson[];      // Array type

      // Note that we use the "IPerson" interface here.
      public addContact(p: IPerson) {
        this.contacts.push(p);
      }

      // A static member...
      static convertToUsd(amountInEur: number): number {
        return amountInEur * 1.3;
      }
    }
  }
}
```

Language Overview

Nested Modules

Note that Person would not need to specify *implements IPerson* explicitly. Even if the *implements* clause would not be there, *Person* would be compatible with *IPerson* because of structural subtyping.

TypeScript Introduction

```
public savePerson(repository, completedCallback: (bool) => void) {
      var code = repository.saveViaRestService(this);
      completedCallback(code === 200);
    }


// Call a method and pass a callback function.
var r = {
  saveViaRestService: function (p: CrmModule.Person) {
    alert("Saving " + p.toString());
    return 200;
  }
};
p.savePerson(r, function(success: string) { alert("Saved"); });
```

Language Overview

Callback functions…

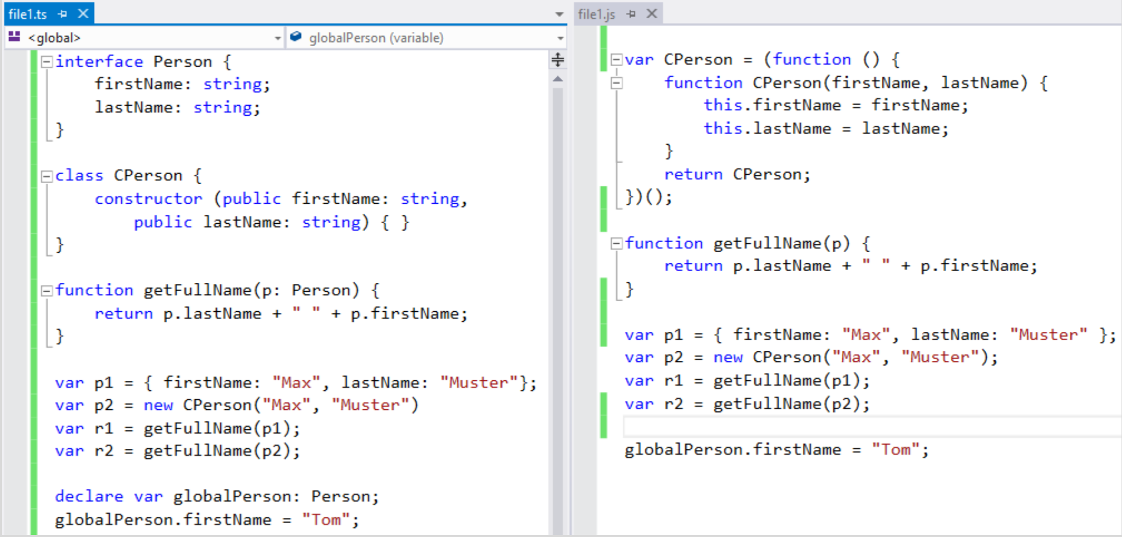
```
export interface IPerson {
  firstName: string;
  lastName: string;
}
…
public addContact(p: IPerson) { this.contacts.push(p); }
…

import S = CrmModule.Sales;
var s: S.Opportunity;
s = new S.Opportunity();
s.potentialRevenueEur = 1000;

s.addContact(v);
s.addContact({ firstName: "Rainer", lastName: "Stropek" });
s.addContact(<CrmModule.IPerson> {
   firstName: "Rainer", lastName: "Stropek" });

var val = S.Opportunity.convertToUsd(s.potentialRevenueEur);
```

Language Overview

Structural Subtyping

Note structural subtyping here. You can call *addContact* with any object type compatible with *IPerson*.

TypeScript Introduction



```
interface Person {
    firstName: string;
    lastName: string;
}

class CPerson {
    constructor (public firstName: string,
        public lastName: string) { }
}

function getFullName(p: Person) {
    return p.lastName + " " + p.firstName;
}

var p1 = { firstName: "Max", lastName: "Muster"};
var p2 = new CPerson("Max", "Muster")
var r1 = getFullName(p1);
var r2 = getFullName(p2);

declare var globalPerson: Person;
globalPerson.firstName = "Tom";
```

```
var CPerson = (function () {
    function CPerson(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    return CPerson;
})();

function getFullName(p) {
    return p.lastName + " " + p.firstName;
}

var p1 = { firstName: "Max", lastName: "Muster" };
var p2 = new CPerson("Max", "Muster");
var r1 = getFullName(p1);
var r2 = getFullName(p2);

globalPerson.firstName = "Tom";
```

What happens with interfaces in JavaScript?

They are gone…

Interfaces

Interfaces are only used for

- editing and compiling
- No type information in resulting JavaScript code

Structural Subtyping

```
interface JQueryEventObject extends Event {
  preventDefault(): any;
}

interface JQuery {
  ready(handler: any): JQuery;
  click(handler: (eventObject: JQueryEventObject) => any): JQuery;
}

interface JQueryStatic {
  (element: Element): JQuery;
  (selector: string, context?: any): JQuery;
}

declare var $: JQueryStatic;
```

Interfaces

Ambient Declarations (*.d.ts*)

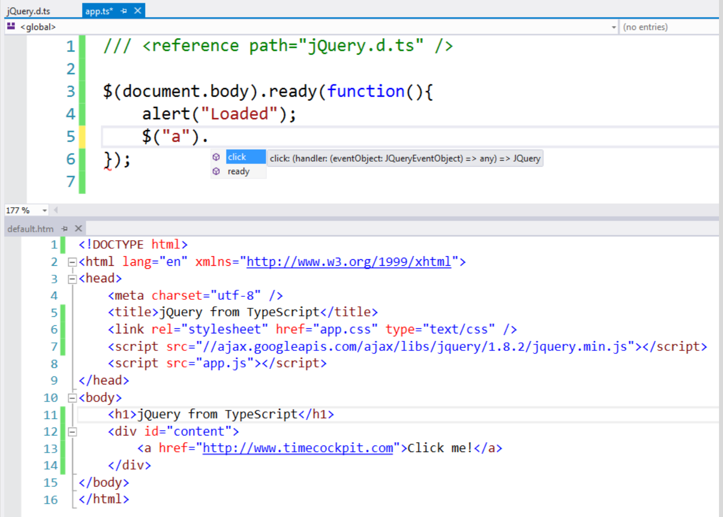
External type information for existing JavaScript libraries like JQuery

TypeScript Type Definition Library

See link in the *resources* section

TypeScript Introduction

```
/// <reference path="jQuery.d.ts" />

$(document.body).ready(function(){
  alert("Loaded");
  $("a").click(function(event) {
    alert("The link no longer took you to timecockpit.com");
    event.preventDefault();
  });
});
```



```
/// <reference path="jQuery.d.ts" />

$(document.body).ready(function(){
    alert("Loaded");
    $("a").
});
```

click: (handler: (eventObject: JQueryEventObject) => any) => JQuery

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>jQuery from TypeScript</title>
    <link rel="stylesheet" href="app.css" type="text/css" />
    <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
    <script src="app.js"></script>
</head>
<body>
    <h1>jQuery from TypeScript</h1>
    <div id="content">
        <a href="http://www.timecockpit.com">Click me!</a>
    </div>
</body>
</html>
```

Interfaces

Ambient Declarations (*.d.ts*)

External type information for existing JavaScript libraries like JQuery

TypeScript Type Definition Library

See link in the *resources* section

TypeScript Introduction

```
export module customer {
    export interface ICustomer {
        firstName: string;
        lastName: string;
    }

    export class Customer implements ICustomer {
        public firstName: string;
        public lastName: string;

        constructor (arg: ICustomer = { firstName: "", lastName: "" }) {
            this.firstName = arg.firstName;
            this.lastName = arg.lastName;
        }

        public fullName() {
            return this.lastName + ", " + this.firstName;
        }
    }
}
```

Shared Code

Common Logic…

- On server (node.js)
- On client (browser)

```
/// <reference path="../tsd/node-0.8.d.ts" />
/// <reference path="../tsd/express-3.0.d.ts" />
/// <reference path="./customer.ts" />
import express = module("express");
import crm = module("customer");

var app = express();

app.get("/customer/:id", function (req, resp) {
    var customerId = <number>req.params.id;
    var c = new crm.customer.Customer({ firstName: "Max" +
customerId.toString(), lastName: "Muster" });
    console.log(c.fullName());
    resp.send(JSON.stringify(c));
});
```

Shared Code

Node.js

Use *express.js* to setup a small web api.

```
app.get("/customer", function (req, resp) {
    var customers: crm.customer.Customer [];
    customers = new Array();
    for (var i = 0; i<10; i++) {
        customers.push(new crm.customer.Customer(
             { firstName: "Max" + i.toString(),
               lastName: "Muster" }));
    }
    resp.send(JSON.stringify(customers));
});

app.use("/static", express.static(__dirname + "/"));

app.listen(8088);
```

Shared Code

Node.js

Use *express.js* to setup a small web api.

Shared Code

Browser

- Uses *require.js* to load modules at runtime

```
/// <reference path="../modules/jquery-1.8.d.ts" />
import cust = module("app/classes/customer");

export class AppMain {
  public run() {
    $.get("http://localhost:8088/Customer/99")
      .done(function (data) {
        var c = new cust.customer.Customer(JSON.parse(data));
        $("#fullname").text(c.fullName());
      });
  }
}
```

So What?

- TypeScript offers you the reach of JavaScript

  Stay as strongly typed as possible but as dynamic as necessary

- TypeScript makes you more productive (IntelliSense)

  Ready for larger projects and larger teams

- TypeScript produces less runtime errors

  Because of compile-time type checking

- TypeScript can change your view on JavaScript

Resources

- ## Videos, Websites, Documents

  http://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript
  http://channel9.msdn.com/posts/Anders-Hejlsberg-Steve-Lucco-and-Luke-Hoban-Inside-TypeScript
  http://www.typescriptlang.org/
  http://www.typescriptlang.org/Playground/
  http://www.typescriptlang.org/Samples/
  http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf

- ## TypeScript Type Definition Library

  https://github.com/borisyankov/DefinitelyTyped

- ## Sample

  http://bit.ly/TypeScriptSample

TypeScript Introduction

Q&A

Thank You For Coming.

Rainer Stropek

software architects gmbh

Mail	rainer@timecockpit.com
Web	http://www.timecockpit.com
Twitter	@rstropek

time cockpit

Saves the day.