TypeScript for JavaScript Programmers

STEVE FENTON

TypeScript for JavaScript Programmers

A fast-track guide to TypeScript

Steve Fenton

This book is for sale at http://leanpub.com/typescriptbook

This version was published on 2013-11-30

Tweet This Book!

Please help Steve Fenton by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought TypeScript for JavaScript Programmers by @steve_fenton!

The suggested hashtag for this book is #typescript.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#typescript

Contents

Introduction

Who this book is for

This book was originally written for JavaScript programmers who are interested in learning TypeScript. The book should actually be useful to anyone who wants to learn TypeScript even if you aren't an expert JavaScript programmer. I have assumed that you know the basics about functions, variables and scope and I won't explain these concepts in this book. I will quickly introduce you to the TypeScript language; explain why it was created, why you would want to use it and, most importantly, how it all works.

What is TypeScript?

Over the past few years there have been many attempts to fix perceived problems with JavaScript. The solutions have been many and varied. There are books listing sub-sets of JavaScript that recommend you don't use any features of the language that aren't listed. There are new languages intended to compile to JavaScript or even replace it in the browser. There was even an attempt to move all interactivity on The Web into proprietary plug-ins that needed to be downloaded for the website to be used. Whether you need one of these solutions very much depends on whether you believe JavaScript is fundamentally broken or not - and I don't think it is. It has its foibles for certain, but JavaScript is one of the most widely adopted languages in the world (TIOBE Programming Community Index for November 2013) and there are good reasons for its success.

TypeScript is a bit different to previous attempts at improving JavaScript because, on the whole, it is just JavaScript with some extra syntactic sugar. Rather than attempting to be the "etch-a-sketch end of the world" for JavaScript, TypeScript is designed to be familiar to those already using JavaScript - and that is a lot of people (75% of web pages in Opera's Metadata Analysis and Mining Application results). It adds some new keywords to help make programming more productive by making the language easier for tools to interpret. This means your development environment can be more intelligent with code completion, static analysis and compile-time warnings.

Some of the language features used in TypeScript are taken from the ECMAScript 6 proposal, which is a new standard for the JavaScript language. Whenever I talk about ECMAScript 6, I am talking about JavaScript from the near future!

The TypeScript language was created by Anders Hejlsberg and a team of developers at Microsoft and is released under an Apache 2 open source license. The preview was released on the same day as the first edition of this book, 1st October 2012, and was labelled version 0.8.

You can follow the project on Codeplex to keep up to date with changes, join in the discussion on the language preview and file any bugs you find while using TypeScript.

Codeplex TypeScript Project[1]

Opera - Metadata Analysis and Mining Application (MAMA)[2]

Code samples

I know that many of the code examples are contrived, but my intention is to use the simplest example possible to demonstrate a given feature. I have also tried to adapt the main example to suit each demonstration so you don't have to work out what I am trying to achieve each time I show off an aspect of TypeScript. I am also aware that code examples never fit onto a page in a book with nice formatting. Please forgive any odd line breaks I have used to squash the examples into the available space. In some cases I have used bad variable names to make the examples fit on the page.

I promise that the real-life code I write is perfectly formatted and my variables are well named. Please don't follow bad practices that I have been forced to apply in order to make code samples fit into the book.

In short

The first edition of this book was designed to be a short introduction to TypeScript that had enough depth to explain the important concepts. TypeScript is still quite new, so you probably just want to get coding. This book will enable you to do that without delving into too much detail. The more complex ideas are later in the book. As the language evolves and matures I will update the book to reflect any changes and also to focus on the areas that crop up most often on support forums.

I have missed out some of the items listed in the language specification, but on the whole I have only done this for features that are already present in JavaScript, such as logical operators, array and object literals and the new keyword.

[1]http://typescript.codeplex.com/

[2]http://dev.opera.com/articles/view/mama/

It starts with JavaScript

To get started with TypeScript, open up your web browser and visit the TypeScript Playground[3]

All of the examples in this book can be tried out in this interactive online development tool. If you prefer to try the examples in a development environment, such as Visual Studio, you can do that too - but using the Playground gives you code-completion, compilation warnings and shows you the resulting JavaScript all in one place, so it is the simplest way to get started.

Let's start out by writing some plain JavaScript into the TypeScript editor. This proves that you can use vanilla JavaScript when writing code in TypeScript and also shows one of the first benefits of TypeScript.

Listing 1.1 – Plain JavaScript

```
function addTwoNumbers() {
    return 10;
}

var result = addTwoNumbers();
```

Listing 1.1 is a very simple example just to prove a point. If you enter this in the TypeScript playground, and hover over **result**, you will see that the type of the variable has been inferred. It is a number. This means the development tool can warn you if you try to do something that doesn't seem sensible with a number. This is because TypeScript is a statically-typed language, whereas JavaScript is dynamically typed. The resulting JavaScript that is created looks identical to the code written in TypeScript - I haven't used any of the new language features yet – but even using plain old JavaScript you can get some type checking for free.

In the following chapters, I will introduce the language features and syntactic sugar that will allow you to specify how your code behaves explicitly, rather than having the TypeScript compiler inferring the behaviour by guessing what you mean. By stating your intentions using these keywords and expressions, you can make your code easier to understand and more predictable to run.

[3]http://www.typescriptlang.org/Playground/

Types

Structural Typing

TypeScript uses structural typing. This means that if two types look the same, they are the same. This is worth bearing in mind if you are used to nominative languages that require you to provide the type name explicitly. In TypeScript, as long as your type has all the required properties and methods declared on an interface or class, you can use that type as if it explicitly implemented the interface or extended the class. This is a very flexible type system.

Type Annotations

Specifying types is very simple in TypeScript. All you need to do is append a type to a variable using a colon to separate the name and type. Let's adjust the example from the previous chapter to make it explicit that **result** should be a number:

Listing 2.1 – Explicit Types

```
function addTwoNumbers() {
    return 10;
}

var result: number = addTwoNumbers();
```

Just as before, hovering over the **result **variable will tell us that it is a number and the compiler will check that I am only using it where it is sensible to use a number – but this time I have made my intentions absolutely clear. If I were to call another method that returned a string, rather than a number the compiler would warn me that there was a problem.

Interesting! Many languages actually put the type before the name of the variable, for example

string name = "name";

The TypeScript team deliberately wanted the type after the variable name to indicate that the type is optional.

It is also possible to specify the type for the parameters in a function declaration. Let's add a real implementation of our **addTwoNumbers** that accepts two numbers and returns the sum of those two numbers.

Listing 2.2 – Typed Parameters

```
function addTwoNumbers(a: number, b: number) {
    return a + b;
}

var result = addTwoNumbers(5, 6);
```

The TypeScript editor will now warn you if you try to pass in an argument that isn't a number. For example, if you were to pass in a string, boolean or KitchenSink. This prevents us from calling the function with arguments that may give us an unexpected result.

Image: Compiler warning of invalid arguments

You can go one step further and specify the return type for the method by adding the type after the brackets:

Listing 2.3 – Return Type

```
function addTwoNumbers(a, b): number {
    return a + b;
}

var result = addTwoNumbers(5, 6);
```

The compiler can now report an error if you try to return the wrong type of response if you change the function body later on and break the expected behaviour.

Primitive types

The primitive types in JavaScript are **string**, **number**, **boolean**, **null** and **undefined**. There are no separate types for integers or numbers with a floating point, they are all **number** types. Later on in the book I will show you how to create your own types, which you can use in just the same way. As well as these simple types, TypeScript has the concept of an **any** type, which is a special type that acts like a plain JavaScript dynamic type. You probably won't use this very often, but it may be useful during the process of converting an existing JavaScript project to TypeScript.

Listing 2.4 – Primitive Types

```
var myBool: boolean = true;

var myString: string = "A String";

var myNumberA: number = 5;

var myNumberB: number = 3.14;

var myAnyA: any = "Another String";

var myAnyB: any = 80;

var myAnyC: any = false;
```

In versions prior to 0.9 of TypeScript, the boolean type was named bool, but this was updated in the early preview of 0.9.

Remember - you don't have to specify the type in cases where it can be inferred by the compiler, but you may wish to make your intentions explicit. Both of the following two statements will behave the same way in TypeScript and the compiler will check the types identically, but in the first example the compiler is inferring the type automatically based on the initial value of the variable:

Listing 2.5 – Implicit vs. Explicit Types

```
var myNumberA = 8; // number

var myNumberB: number = 16; // number
```

If you are not setting the variable, but simply declaring it for later use, I recommend that you explicitly state the type, otherwise the compiler will assume the type is **any**.

Listing 2.6 – Declaring Variables with No Value

```
var myVarA; // any

var myVarB: number; // number, with a value of undefined
```

It is a matter of personal choice - or team choice if you are working with other programmers - whether you explicitly state the type of a variable in all cases. I recommend that you discuss the options and agree on the style you find most readable.

Because TypeScript adds these types as keywords, you should add them to the list of reserved words and you should avoid using them in your own code, for example you wouldn't want to name a variable **string**:

Listing 2.7 – Using Reserved Words as Variable Names

```
var string: string = "";
```

Object types

Object types include classes, modules and interfaces and are custom types that you create to represent concepts in your code. You might use them to store a number of some simple types, like a data contract, or you may include behaviour as well as data. We will look at them in more detail later in the book.