1 Introduction

JavaScript applications such as web e-mail, maps, document editing, and collaboration tools are becoming an increasingly important part of the everyday computing. We designed TypeScript to meet the needs of the JavaScript programming teams that build and maintain large JavaScript programs. TypeScript helps programming teams to define interfaces between software components and to gain insight into the behavior of existing JavaScript libraries. TypeScript also enables teams to reduce naming conflicts by organizing their code into dynamically-loadable modules. TypeScript's optional type system enables JavaScript programmers to use highly-productive development tools and practices: static checking, symbol-based navigation, statement completion, and code re-factoring.

TypeScript is a syntactic sugar for JavaScript. TypeScript syntax is a superset of Ecmascript 5 (ES5) syntax. Every JavaScript program is also a TypeScript program. The TypeScript compiler performs only file-local transformations on TypeScript programs and does not re-order variables declared in TypeScript. This leads to JavaScript output that closely matches the TypeScript input. TypeScript does not transform variable names, making tractable the direct debugging of emitted JavaScript. TypeScript optionally provides source maps, enabling source-level debugging. TypeScript tools typically emit JavaScript upon file save, preserving the test, edit, refresh cycle commonly used in JavaScript development.

TypeScript syntax includes several proposed features of Ecmascript 6 (ES6), including classes and modules. Classes enable programmers to express common object-oriented patterns in a standard way, making features like inheritance more readable and interoperable. Modules enable programmers to organize their code into components while avoiding naming conflicts. The TypeScript compiler provides module code generation options that support either static or dynamic loading of module contents.

TypeScript also provides to JavaScript programmers a system of optional type annotations. These type annotations are like the JSDoc comments found in the Closure system, but in TypeScript they are integrated directly into the language syntax. This integration makes the code more readable and reduces the maintenance cost of synchronizing type annotations with their corresponding variables.

The TypeScript type system enables programmers to express limits on the capabilities of JavaScript objects, and to use tools that enforce these limits. To minimize the number of annotations needed for tools to become useful, the TypeScript type system makes extensive use of type inference. For example, from the following statement, TypeScript will infer that the variable 'i' has the type number.

```
var i = 0;
```

TypeScript will infer from the following function definition that the function f has return type string.

```
function f() {
    return "hello";
}
```