1.1 Ambient Declarations

An ambient declaration introduces a variable into a TypeScript scope, but has zero impact on the emitted JavaScript program. Programmers can use ambient declarations to tell the TypeScript compiler that some other component will supply a variable. For example, by default the TypeScript compiler will print an error for uses of undefined variables. To add some of the common variables defined by browsers, a TypeScript programmer can use ambient declarations. The following example declares the 'document' object supplied by browsers. Because the declaration does not specify a type, the type 'any' is inferred. The type 'any' means that a tool can assume nothing about the shape or behavior of the document object. Some of the examples below will illustrate how programmers can use types to further characterize the expected behavior of an object.

```
declare var document;
document.title = "Hello";  // Ok because document has been declared
```

In the case of 'document', the TypeScript compiler automatically supplies a declaration, because TypeScript by default includes a file 'lib.d.ts' that provides interface declarations for the built-in JavaScript library as well as the Document Object Model.

The TypeScript compiler does not include by default an interface for jQuery, so to use jQuery, a programmer could supply a declaration such as:

```
declare var $;
```

Section 1.3 provides a more extensive example of how a programmer can add type information for jQuery and other libraries.

1.2 Function Types

Function expressions are a powerful feature of JavaScript. They enable function definitions to create closures: functions that capture information from the lexical scope surrounding the function's definition. Closures are currently JavaScript's only way of enforcing data encapsulation. By capturing and using environment variables, a closure can retain information that cannot be accessed from outside the closure. JavaScript programmers often use closures to express event handlers and other asynchronous callbacks, in which another software component, such as the DOM, will call back into JavaScript through a handler function.

TypeScript function types make it possible for programmers to express the expected *signature* of a function. A function signature is a sequence of parameter types plus a return type. The following example uses function types to express the callback signature requirements of an asynchronous voting mechanism.

```
function vote(candidate: string, callback: (result: string) => any) {
   // ...
}
```