In this example, variable ‘s’ is a private feature of the module, but function ‘f’ is exported from the module and accessible to code outside of the module. If we were to describe the effect of module ‘M’ in terms of interfaces and variables, we would write

```
interface M {
    f(): string;
}

var M: M;
```

The interface ‘M’ summarizes the externally visible behavior of module ‘M’. In this example, we can use the same name for the interface as for the initialized variable because in TypeScript type names and variable names do not conflict: each lexical scope contains a variable declaration space and type declaration space (see section 2.3 for more details).

Module ‘M’ is an example of an *internal* module, because it is nested within the *global* module (see section 10 for more details). The TypeScript compiler emits the following JavaScript code for this module.

```
var M;
(function(M) {
    var s = "hello";
    function f() {
        return s;
    }
    M.f = f;
})(M||(M={}));
```

In this case, the compiler assumes that the module object resides in global variable ‘M’, which may or may not have been initialized to the desired module object.

TypeScript also supports *external* modules, which are files that contain top-level *export* and *import* directives. For this type of module the TypeScript compiler will emit code whose module closure and module object implementation vary according to the specified dynamic loading system, for example, the Asynchronous Module Definition system.