10 Internal Modules

An internal module is a named container of statements and declarations. An internal module represents both a namespace and a singleton module instance. The namespace contains named types and other namespaces, and the singleton module instance contains properties for the module's exported members. The body of an internal module corresponds to a function that is executed once, thereby providing a mechanism for maintaining local state with assured isolation.

10.1 Module Declarations

An internal module declaration declares a namespace name and, in the case of an instantiated module, a member name in the containing module.

ModuleDeclaration:
`module` *IdentifierPath* `{` *ModuleBody* `}`

IdentifierPath:
Identifier
IdentifierPath `.` *Identifier*

Internal modules are either ***instantiated*** or ***non-instantiated***. A non-instantiated module is an internal module containing only interface types and other non-instantiated modules. An instantiated module is an internal module that doesn't meet this definition. In intuitive terms, an instantiated module is one for which a module object instance is created, whereas a non-instantiated module is one for which no code is generated.

When a module identifier is referenced as a *ModuleName* (section 3.6.2) it denotes a container of module and type names, and when a module identifier is referenced as a *PrimaryExpression* (section 4.3) it denotes the singleton module instance. For example:

```
module M {
    export interface P { x: number; y: number; }
    export var a = 1;
}

var p: M.P;             // M used as ModuleName
var m = M;              // M used as PrimaryExpression
var x1 = M.a;           // M used as PrimaryExpression
var x2 = m.a;           // Same as M.a
var q: m.P;             // Error
```

Above, when 'M' is used as a *PrimaryExpression* it denotes an object instance with a single member 'a' and when 'M' is used as a *ModuleName* it denotes a container with a single type member 'P'. The final line in the example is an error because 'm' is a variable which cannot be referenced in a type name.