is possible to add properties to an object after its creation, it is not possible to make an object "callable" after the fact.)

The example

```
interface Point {
    x: number;
    y: number;
}

function point(x: number, y: number): Point {
    return { x: x, y: y };
}

module point {
    export var origin = point(0, 0);
    export function equals(p1: Point, p2: Point) {
        return p1.x == p2.x && p1.y == p2.y;
    }
}

var p1 = point(0, 0);
var p2 = point.origin;
var b = point.equals(p1, p2);
```

declares 'point' as a function object with two properties, 'origin' and 'equals'. Note that the module declaration for 'point' is located after the function declaration.

10.6 Code Generation

An internal module generates JavaScript code that is equivalent to the following:

```
var <ModuleName>;
(function(<ModuleName>) {
    <ModuleStatements>
})(<ModuleName>||(<ModuleName>={}));
```

where *ModuleName* is the name of the module and *ModuleStatements* is the code generated for the statements in the module body. The *ModuleName* function parameter may be prefixed with one or more underscore characters to ensure the name is unique within the function body. Note that the entire module is emitted as an anonymous function that is immediately executed. This ensures that local variables are in their own lexical environment isolated from the surrounding context. Also note that the generated function doesn't create and return a module instance, but rather it extends the existing instance (which may have just been created in the function call). This ensures that internal modules can extend each other.

An import statement generates code of the form

```
var <Alias> = <EntityName>;
```