Top-level declarations in a source file with no top-level import or export declarations belong to the ***global module***. Top-level declarations in a source file with one or more top-level import or export declarations belong to the ***external module*** represented by that source file.

An internal module declaration contributes a namespace name (representing a container of types) and possibly a member name (representing the module instance) to the containing module. A class declaration contributes both a member name (representing the constructor function) and a type name (representing the class type) to the containing module. An interface declaration contributes a type name to the containing module. An enum declaration contributes both a member name (representing the enum object) and a type name (representing the enum type) to the containing module. Any other declaration contributes a member name to the declaration space to which it belongs.

The ***parent module*** of an entity is defined as follows:

- The parent module of an entity declared in an internal module is that internal module.
- The parent module of an entity declared in an external module is that external module.
- The parent module of an entity declared in the global module is the global module.
- The parent module of an external module is the global module.

The ***root module*** of an entity is defined as follows:

- The root module of a non-exported entity is the entity's parent module.
- The root module of an exported entity is the root module of the entity's parent module.

Intuitively, the root module of an entity is the outermost module body from within which the entity is reachable.

Interfaces, enums, and internal modules are "open ended," meaning that interface, enum, and internal module declarations with the same qualified name relative to a common root are automatically merged. For further details, see sections 7.2, 9.3, and 10.5.

Namespace, type, and member names exist in separate declaration spaces. Furthermore, declarations of non-instantiated modules (modules that contain only interfaces or modules at all levels of nesting) do not introduce a member name in their containing declaration space. This means that the following is permitted, provided module 'X' contains only interface or module declarations at all levels of nesting:

```
module M {
    module X { ... }      // Namespace
    interface X { ... }   // Type
    var X;                // Member
}
```

If module 'X' above was an instantiated module (section 10.1) it would cause a member 'X' to be introduced in 'M'. This member would conflict with the variable 'X' and thus cause an error.