# Changes to 13.2.1 (“Let and Const Declarations”)

**Syntax**

*BindingIdentifier* :

*Identifier*

**default**

**Static Semantics: Early Errors**

*BindingIdentifier* : **default**

* It is a Syntax Error if the innermost *Declaration* or *VariableStatement* containing this production is not immediately contained in an *ExportDeclaration*.

# Changes to 15.1 (“Script”)

**Syntax**

*Script* :

*ScriptBodyopt*

*ScriptBody* :

*ScriptOuterItemList*

*ScriptOuterItemList* :

*ScriptOuterItem*

*ScriptOuterItemList* *ScriptOuterItem*

*ScriptOuterItem* :

*ModuleDeclaration*

*ImportDeclaration*

*StatementListItem*

# Changes to 15.3 (“Modules”)

**Syntax**

*Module* :

*ModuleBodyopt*

*ModuleBody* :

*ModuleOuterItemList*

*ModuleOuterItemList* :

*ModuleOuterItem*

*ModuleOuterItemList* *ModuleOuterItem*

*ModuleOuterItem* :

*ModuleDeclaration*

*ImportDeclaration*

*ExportDeclaration*

*StatementListItem*

*ModuleDeclaration* :

**module** [no *LineTerminator* here] *Identifier* **from** *ModuleSpecifier* **;**

*ModuleSpecifier* :

*StringLiteral*

*ImportDeclaration* :

**import** *ImportSpecifierSet* **from** *ModuleSpecifier* ;

**import** *ModuleSpecifier* **;**

*ImportSpecifierSet* :

*Identifier*

**{** **}**

**{** *ImportSpecifierList* **}**

**{** *ImportSpecifierList* **,** **}**

*ImportSpecifierList* :

*ImportSpecifier*

*ImportSpecifierList* **,** *ImportSpecifier*

*ImportSpecifier* :

*Identifier*

*Identifier* **as** *Identifier*

*ReservedWord* **as** *Identifier*

*ExportDeclaration* :

**export** *ExportSpecifierSet* **;**

**export** *ExportSpecifierSet* **from** *ModuleSpecifier* **;**

**export** *VariableStatement* **;**

**export** *Declaration* **;**

**export** *BindingList* **;**

*ExportSpecifierSet* :

*ExplictExportSpecifierSet*

**\***

*ExplicitExportSpecifierSet* :

**{** **}**

**{** *ExportSpecifierList* **}**

**{** *ExportSpecifierList* **,** **}**

*ExportSpecifierList* :

*ExportSpecifier*

*ExportSpecifierList* **,** *ExportSpecifier*

*ExportSpecifier* :

*Identifier*

*Identifier* **as** *IdentifierName*

**Static Semantics: Early Errors**

*Script* : *ScriptBodyopt*

* It is a Syntax Error if the ImportedNames of *ScriptBodyopt* contains any duplicate elements.

*Module* : *ModuleBodyopt*

* It is a Syntax Error if the ImportedNames of *ModuleBodyopt* contains any duplicate elements.

*ImportDeclaration* : **import** *ImportSpecifierSet* **from** *ModuleSpecifier* **;**

* It is a Syntax Error if the BoundNames of *ImportSpecifierSet* contains the string “**eval**” or the string “**arguments**”.

**Static Semantics: BoundNames**

*ImportDeclaration* : **import** *ImportSpecifierSet* **from** *ModuleSpecifier* **;**

1. Let *names* be the ImportedNames of *ImportSpecifierSet*.
2. Return *names*.

*ImportDeclaration* : **import** *ModuleSpecifier* **;**

1. Return a new empty List.

*ExportDeclaration* : **export** *VariableStatement* **;**

1. Let *names* be the BoundNames of *VariableStatement*.
2. Return *names*.

*ExportDeclaration* : **export** *Declaration* **;**

1. Let *names* be the BoundNames of *Declaration*.
2. Return *names*.

*ExportDeclaration* : **export** *BindingList* **;**

1. Let *names* be the BoundNames of *BindingList*.
2. Return *names*.

*ModuleDeclaration* : **module** [no *LineTerminator* here] *Identifier* **from** *ModuleSpecifier* **;**

1. Let *names* be a new List containing *Identifier*.
2. Return *names*.

**Static Semantics: ImportedNames**

*ImportDeclaration* : **import** *ImportSpecifierSet* **from** *ModuleSpecifier* **;**

1. Return the ImportedNames of *ImportSpecifierSet*.

*ImportSpecifierSet* : *Identifier*

1. Return a new List containing *Identifier*.

*ImportSpecifierSet* : **{** **}**

1. Return a new empty List.

*ImportSpecifierSet* : **{** *ImportSpecifierList* **}**

*ImportSpecifierSet* : **{** *ImportSpecifierList* **,** **}**

1. Return the ImportedNames of *ImportSpecifierList*.

*ImportSpecifierList* : *ImportSpecifier*

1. Return the ImportedNames of *ImportSpecifier*.

*ImportSpecifierList* : *ImportSpecifierList* **,** *ImportSpecifier*

1. Let *names* be the ImportedNames of *ImportSpecifierList*.
2. Append to *names* the elements of the ImportedNames of *ImportSpecifier*.
3. Return *names*.

*ImportSpecifier* : *Identifier*

1. Return a new List containing *Identifier*.

*ImportSpecifier* : *Identifier* **as** *Identifier*

1. Return a new List containing the second *Identifier*.

*ImportSpecifier* : *ReservedWord* **as** *Identifier*

1. Return a new List containing *Identifier*.

*ImportDeclaration* : **import** *ModuleSpecifier* **;**

1. Return a new empty List.

**Static Semantics: ExportedNames**

With arguments *mod* and *visited*.

*ExportDeclaration* : **export** **\*;**

1. Let *body* be *mod*.[[Body]].
2. Let *names* be the BoundNames of *body*.
3. Let *importedNames* be the ImportedNames of *body*.
4. Let *localNames* be the set difference of *names* and *importedNames*.
5. Return *localNames*.

*ExportDeclaration* : **export** *ExplicitExportSpecifierSet* **;**

1. Let *body* be *mod*.[[Body]].
2. Let *names* be the BoundNames of *body*.
3. Let *exports* be a new empty List.
4. Let *pairs* be the ExportPairs of *ExplicitExportSpecifierSet*.
5. For each *pair* in *pairs*, do:
   1. Let *i* be *pair*.[[Internal]].
   2. If *i* is not in *names*, throw a new **ReferenceError**.
   3. Let *e* be *pair*.[[External]].
   4. Add *e* to *exports*.
6. Return *exports*.

*ExportDeclaration* : **export** **\*****from** *ModuleSpecifier* **;**

1. Let *deps* be *mod*.[[Dependencies]].
2. Let *r* be the record in *deps* such that *r*.[[Name]] equals *ModuleSpecifier*.
3. Let *otherMod* be *r*.[[Module]].
4. If *otherMod* is in *visited*, throw a new **SyntaxError**.
5. Add *otherMod* to *visited*.
6. Let *exports* be the result of calling the ModuleInstanceExportedNames abstract operation with arguments *otherMod* and *visited*.
7. ReturnIfAbrupt(*exports*).
8. Return *exports*.

*ExportDeclaration* : **export** *ExplicitExportSpecifierSet* **from** *ModuleSpecifier* **;**

1. Let *pairs* be the ExportPairs of *ExportSpecifierSet*.
2. Let *exports* be a new empty List.
3. For each *pair* in *pairs*, do:
   1. Let *e* be *pair*.[[External]].
   2. Add *e* to *exports*.
4. Return *exports*.

*ExportDeclaration* : **export** *VariableStatement* **;**

1. Let *names* be the BoundNames of *VariableStatement*.
2. Return *names*.

*ExportDeclaration* : **export** *Declaration* **;**

1. Let *names* be the BoundNames of *Declaration*.
2. Return *names*.

*ExportDeclaration* : **export** *BindingList* **;**

1. Let *names* be the BoundNames of *BindingList*.
2. Return *names*.

**Static Semantics: ExportPairs**

*ExplicitExportSpecifierSet* : **{** **}**

1. Return a new empty List.

*ExplicitExportSpecifierSet* : **{** *ExportSpecifierList* **}**

*ExplicitExportSpecifierSet* : **{** *ExportSpecifierList* **,** **}**

1. Let *pairs* be the ExportPairs of *ExportSpecifierList*.
2. Return *pairs.*

*ExportSpecifierList* : *ExportSpecifier*

1. Let *pairs* be the ExportPairs of *ExportSpecifier*.
2. Return *pairs*.

*ExportSpecifierList* : *ExportSpecifierList* **,** *ExportSpecifier*

1. Let *pairs* be the ExportPairs of *ExportSpecifierList*.
2. Append the ExportPairs of *ExportSpecifier* to *pairs*.
3. Return *pairs*.

*ExportSpecifier* : *Identifier*

1. Let *pairs* be a new List containing the record {[[Internal]]: *Identifier*, [[External]]: *Identifier*}.
2. Return *pairs*.

*ExportSpecifier* : *Identifier* **as** *IdentifierName*

1. Let *pairs* be a new List containing the record {[[Internal]]: *Identifier*, [[External]]: *IdentifierName*}.
2. Return *pairs*.

## New section: Modules and Module Loaders

**ModuleInstanceExportedNames ( M, visited )**

When the abstract operation ModuleInstanceExportedNames is called with arguments *M* and *visited*, the following steps are taken:

1. If *M*.[[ExportedNames]] is not **undefined**, then return *M*.[[ExportedNames]].
2. Let *body* be the *Module* parse stored at *M*.[[Body]].
3. Let *exports* be the ExportedNames of *body* passing *M* and *visited*.
4. Set *M*.[[ExportedNames]] to *exports*.
5. Return *exports*.

**DefineBuiltinProperties ( O )**

When the abstract operation DefineBuiltinProperties is called with object *O*, the following steps are taken:

1. For each name *P* of the standard properties of the global object (see 18), in alphabetical order, do:
   1. Let *V* be the value of the standard built-in property *P* for the current realm.
   2. Let *W* be the value of the [[Writable]] attribute of the standard built-in property *P*.
   3. Let *E* be the value of the [[Enumerable]] attribute of the standard built-in property *P*.
   4. Let *C* be the value of the [[Configurable]] attribute of the standard built-in property *P*.
   5. Let *newDesc* be the Property Descriptor {[[Value]]: *V*, [[Writable]]: *W*, [[Enumerable]]: *E*, [[Configurable]]: *C*}.
   6. Let *status* be the result of calling the [[DefineOwnProperty]] internal method of *O* passing *P* and *newDesc* as arguments.
   7. ReturnIfAbrupt(*status*).
2. Return *O*.

# Changes to 15.1.2 (“Script Evaluation”)

**Runtime Semantics: Script Evaluation**

*Script* : *ScriptBodyopt*

With arguments *realm*, *deletableBindings*, and *moduleDependencies*.

(before the existing step 1)

1. For each *mod* in *moduleDependencies*, do:
   1. Let *status* be the result of performing Module Evaluation of *mod* using *realm*, *deletableBindings*, and *mod*.[[Dependencies]] as arguments.
   2. ReturnIfAbrupt(*status*).