orphan:

> **Warning**
>
> This document represents an early proposal for `import` syntax and has not been kept up to date.

IMPORT SYNTAX

```
import-decl ::= 'import' import-item-list
import-item-list ::= import-item (',' import-item)*

import-item ::= import-kind? identifier-path
import-item ::= identifier-path '.' '(' import-item-list ')'

import-kind ::= 'module'

import-kind ::= 'class'
import-kind ::= 'enum'
import-kind ::= 'func'
import-kind ::= 'protocol'
import-kind ::= 'struct'
import-kind ::= 'typealias'
import-kind ::= 'var'
// ...
```

`import` makes declarations exported from another module available inside the current module. Imports are not reexported by default.

Importing Modules

In its simplest form, `import` gives the qualified name of a module and imports all exported symbols from the module, as well as the module name itself for qualified lookup:

```
import Cocoa

// Reference the NSArray type from Cocoa
var a1 : NSArray
// Same, but qualified
var a2 : Cocoa.NSArray
```

In this form, the qualified name *must* refer to a module:

```
// Import the Cocoa.NSWindow module, *not* the NSWindow class from inside
// Cocoa
import Cocoa.NSWindow

// Reference the NSWindow type from Cocoa.NSWindow
var w1 : NSWindow
// Same, but qualified
var w2 : Cocoa.NSWindow.NSWindow
```

Multiple modules may appear in a comma-separated list:

```
import Foundation, iAd, CoreGraphics
```

As a shorthand, multiple submodules with a common parent module may be listed in parens under the parent module:

```
import OpenGL.(GL3, GL3.Ext)
```

Importing Individual Declarations

Instead of importing the entire contents of a module, individual declarations may be imported. This is done by naming the kind of declaration being imported before the qualified name, such as `func`, `var`, or `class`. The module name is still imported for qualified lookup of other symbols:

```
// Import only the Cocoa.NSWindow class
import class Cocoa.NSWindow

var w1 : NSWindow
var title : Cocoa.NSString
```

As with modules, multiple declarations may be imported in a comma-separated list, or imported out of a common parent module with a parenthesized list:

```
import func OpenGL.GL3.glDrawArrays, func OpenGL.GL3.Ext.glTextureRangeAPPLE
// Equivalent
import OpenGL.GL3.(func glDrawArrays, func Ext.glTextureRangeAPPLE)
```

RESOLVING NAME CLASHES

Module imports

Because the local names introduced by a whole-module import are implicit, a name clash between imported modules is not an error unless a clashing name is actually used without qualification:

```
import abcde // abcde exports A, B, C, D, E
import aeiou // aeiou exports A, E, I, O, U

var b : B // OK, references abcde.B
var i : I // OK, references aeiou.I
var e : E // Error, ambiguous
var e : abcde.E // OK, qualified reference to abcde.E
```

Conflicts are resolved in favor of individually imported or locally defined declarations when available:

```
import abcde         // abcde exports A, B, C, D, E
import aeiou         // aeiou exports A, E, I, O, U
import class asdf.A  // explicitly import A from some other module
import class abcde.E // explicitly import E from abcde

class U { } // Local class shadows whole-module import

var a : A // OK, references asdf.A
var e : E // OK, references abcde.E
var u : U // OK, references local U
```

Declaration imports

Individual declaration imports shadow whole-module imports, as described above. If two declarations with the same name are individually imported from different modules, references to either import must be qualified:

```
import class abcde.E
import class aeiou.E

var e : E        // Error, ambiguous
var e1 : abcde.E // OK
```

A local definition with the same name as an explicitly imported symbol shadows the unqualified import:

```
import class abcde.E

class E { }

var e : E       // Refers to local E
var e : abcde.E // Refers to abcde.E
```

Module names

FIXME: What is a good rule here? This sucks.

If a module name clashes with a local definition or imported declaration, the declaration is favored in name lookup. If a member lookup into the declaration fails, we fall back to qualified lookup into the module:

```
import Foo // exports bas

class Foo {
  class func bar()
}

Foo.bar() // bar method from Foo class
Foo.bas() // bas method from Foo module
```

FUTURE EXTENSIONS

In the future, we should allow the import declaration to provide an alias for the imported module or declaration:

```
import C = Cocoa
import NSW = class Cocoa.NSWindow
import Cocoa.(NSW = class NSWindow, NSV = class NSView)
```