

# Structure of a Boo Script

Boo scripts have a set structure you need to follow to avoid errors when compiling. All of the items are optional, but order of the items is not:

The structure is like:

- module docstring
- namespace declaration
- import statements
- module members: class/enum/def declarations
- main code executed when script is run
- assembly attributes

For example:

```

"""module docstring"""

namespace My.NameSpace #optional namespace
declaration

import Assembly.Reference #import statements

#followed by the Members of this module
(classes, methods, etc.)
class MyClass:
    pass

def domyfunction(it):
    print(it)

#start "main" section that is executed when
script is run
x as int
x = 3
domyfunction(x)

#optional assembly attribute declarations
used when compiling
[assembly: AssemblyTitle('foo')]
[assembly: AssemblyDescription('bar')]

```

---

There is one issue with the above structure, however. What if you are creating a library (dll), and you want to have "global" properties or fields? Also, what if you want to do some things when your dll is loaded by another application?

Example use:

```
import MyLibrary
print (Version)
doit()
```

Now in Boo, you can do this by specifying a "global" class for your module. You tell the Boo compiler to use your class as the global, module-wide class by adding a [Module] attribute:

```
[Module]
class MainClass:
    public static Version as string

    static def constructor():
        Version = "0.1"

def doit():
    #you can refer to "globals" from within
    your library, too:
    print("This library's version is:
    "+MainClass.Version)
```

---

Also in Boo you can define which method the compiler should use as the main entry point method. This is useful for example when you want to add an attribute (like STAThread) to the main method. This main method can be anywhere in your boo script, or you can put it inside your main module class if you want to combine both of these techniques.

```
[STAThread]
def Main(argv as (string)):
    ...main entry point
```