

Scripting with the Boo.Lang.Compiler API

This is actually relatively simple; we only need a few of the Boo.Lang.Compiler classes to make it happen.

The classes most important to us are:

- Boo.Lang.Compiler.BooCompiler - the class that handles compilation of Boo code.
- Boo.Lang.Compiler.Pipelines.CompileToMemory - a compiler pipeline that compiles Boo code into memory instead of a file output. You'll want this for most scripting situations.

Using these classes and reflection, we will be able compile Boo scripts to memory and run'em on inputted variables.

Suppose script.boo is a class that the user has written, and you want to consume it by your application without the user having to dance with booc.exe or some other fancy pants method.

script.boo

```
static def stringManip(item as string):  
    //static lets us invoke this method without  
    needing to instanize a class.  
    return "'${item}'? What the hell are you  
    talking about?"
```

Calling scripting Boo from Boo is ridiculously easy--too easy to even explain, so here is the commented code.

runBoo.boo

```
import System  
import Boo.Lang.Compiler  
import Boo.Lang.Compiler.IO  
import Boo.Lang.Compiler.Pipelines  
  
booC = BooCompiler()  
booC.Parameters.Input.Add(  
    FileInput("script.boo") )  
booC.Parameters.Pipeline = CompileToMemory()  
//No need for an on-disk file.  
booC.Parameters.Ducky = true //By default,
```

all objects will be ducked typed; no need for the user to "var as string" anywhere.

```
context = booC.Run()  
//The main module name is always  
filename+Module in pascal case;  
//this file is actually RunBooModule!  
//Using duck-typing, we can directly invoke  
static methods  
//Without having to do the typical  
System.Reflection voodoo.  
if context.GeneratedAssembly is null:  
    print join(e for e in context.Errors, "\n")  
else:  
    var as duck =  
context.GeneratedAssembly.GetType("ScriptMod  
ule")  
  
print var.stringManip("Dance Dance
```

```
Revolution")
    print var.stringManip("Techno music gives
me hives.")
```

That was painless, wasn't it? As you're imaging, you can instanize classes and call instance methods via the convience of duck typing in a similar fashion. Consider the code-block below.

```
myClass =
context.GeneratedAssembly.GetType("SomeClass
", true, true) //Here, we need the type.
myInstance as duck = myClass() //Create an
instanization of this type!
myInstance.myMethod("Happy happy joy joy!")
```

That's it. Use myInstance as you would any other class, **sans code completion** ⚠

This method, of course, is not as interesting as using another language entirely to invoke our Boo script. Let's see how we do it in C#:

runBoo.cs

```
using System;
using System.Text;
using System.Reflection;

using Boo.Lang.Compiler;
using Boo.Lang.Compiler.IO;
using Boo.Lang.Compiler.Pipelines;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
```

```
    {
        BooCompiler compiler = new
BooCompiler();

compiler.Parameters.Input.Add(new
FileInput("script.boo"));
        compiler.Parameters.Pipeline =
new CompileToMemory();
        compiler.Parameters.Ducky =
true;

        CompilerContext context =
compiler.Run();
        //Note that the following code
might throw an error if the Boo script had
bugs.

        //Poke context.Errors to make
sure.

        if (context.GeneratedAssembly !=
null)
        {
            Type scriptModule =
context.GeneratedAssembly.GetType("ScriptMod
ule");

            MethodInfo stringManip =
scriptModule.GetMethod("stringManip");
            string output =
(string)stringManip.Invoke(null, new
object[] { "Tag" } );
            Console.WriteLine(output);
        }
    }
```

```
        else
        {
            foreach (CompilerError error
in context.Errors)
Console.WriteLine(error);
        }
```

```
}  
    }  
}
```

I compiled the above runBoo.cs script with this command (you can change csc to gmcs if you use mono):

```
csc /r:Boo.Lang.Compiler.dll runBoo.cs
```

If you are on Windows, make sure you specify the paths to the boo dlls. After compiling, either move runBoo.exe into the same folder as the boo dlls or copy the dlls to the same folder as your exe. Then run the exe, making sure "script.boo" is in the same folder as your current directory:

```
runBoo.exe
```

As you can see the C# variant is a bit more verbose, but that's your bag, let it roll.

Here are the highlights of the C# version.

```
Type scriptModule =  
context.GeneratedAssembly.GetType("ScriptMod  
ule");
```

ScriptModule is the name of the class encapsulating the main method and the stringManip method of the compiled "script.boo" It is a uniquely generated name determined by the file-name plus a "Module" postfix.

```
string output =  
(string)stringManip.Invoke(null, new  
object[] { "Tag" } );
```

This block of code invokes the stringManip method with one parameter: "Tag." The first parameter, which is normally an instance of a class, is null because the stringManip method is static and thus requires no instance method of ScriptModule.

It returns back the output of the stringManip method.

