Boo Unit Testing

Home Up

Why use Boo for Unit Testing your C# projects?

Given that, just like in the java/jython world, many projects might still want to write their main app code in C# and all their unit tests in Boo - we should help people with this. In fact this might be an important key to the wider acceptance of Boo within the .NET community.

How to do it

I ran into a number of unresolved issues trying to get unit testing happening with Boo, but its was probably my inexperience - since I believe there are unit tests already for testing boo itself. However these unit tests might be written in C#? I'm talking about how to write unit test in Boo itself, for testing both Boo code and C# code.

There is some info at http://boo.codehaus.org/Writing+Tests about unit testing and Boo.

update - Sep 2005

You *can* write boo unit tests that are runnable within nUnit Gui and from the nunit console. See

http://boo.codehaus.org/Unit+Testing+and+You%2C+a+guide

See also my tips for Developing & Debugging visually in Boo

THIS IS SOMEWHAT OUT OF DATE SINCE **I CAN NOW WRITE UNIT TEST IN BOO SUCCESSFULLY** - BUT THIS MAY BE OF SOME USE TO KEEP THIS INFO HERE.

Basically, I cannot do what I want yet - here is where I got up to though:

Original Post:

I really want to be able to build some nunit tests for my boo experiments and want to avoid learning nant and delving into the innards of the way unit tests for boo itself are constructed. I just want to get a simple unit test class working from the command line.

This is the boo source code containing a unit test written in Boo. I want this to run and invoke the unittesting framework and run the tests:

```
------------------ "andy08 - nunit03.boo" -----
import System
import NUnit.Framework

[TestFixture]
class ATestCase:
      _list as List

     [SetUp]
     def SetUp():
       _list = ["um", "dois", "tres"]

     [Test]
     def Slicing():
       Assert.AreEqual(["um"], _list[:1])
       Assert.AreEqual(["um", "dois"], _list[:2])

     [Test]
     def ShouldFail():
       Assert.AreEqual(["um"], ["zzzzzzz"])
```

And my attempted commands:

Using **booi** I get a path not found,

```
C:\boo\examples>..\bin\booi.exe -r:"C:\Program Files\NUnit 2.2\bin\nunit.framework.dll" "andy08 - nunit03.boo"

BCE0056: Boo.Lang.Compiler.CompilerError: File '-r:C:\Program Files\NUnit 2.2\bin\nunit.framework.dll' was not found.
at Boo.Lang.Compiler.IO.FileInput.Open()
at Boo.AntlrParser.BooParsingStep.Run()
```

and using **booc** I get a No entry point found error.

```
C:\boo\examples>..\bin\booc.exe -r:"C:\Program Files\NUnit 2.2\bin\nunit.framework.dll" "andy08 - nunit03.boo"

BCE0028: No entry point found.
1 error(s).
```

Some limited success !!

On the other hand I did have some limited success with the following script

```
------------- "andy08 - nunit01.boo" --------
import NUnit.Framework

class Foo:

    _prefix

    def constructor(prefix):
        _prefix = prefix

    def call(c as ICallable):
        return c()

    def foo():
        return "${_prefix} - foo"

    def run():
        return call(foo)

Assert.AreEqual("bar - foo", Foo("bar").run())  // should pass
Assert.AreEqual("bar - foo", "zzzzz")  // should fail
```

As usual, booi didn't work and gave me the same file not found error as above.

However booc *did* work (which incidently proves that my path reference to nunit framework dll DOES exist ;-)

C:\boo\examples>..\bin\booc.exe -r:"C:\Program Files\NUnit 2.2\bin\nunit.framework.dll" "andy08 - nunit01.boo"

And when I run it I get one pass and one failure, just as I expected :-)

```
C:\boo\ANDYDE~1>"andy08 - nunit01.exe"

Unhandled Exception: NUnit.Framework.AssertionException:
String lengths differ. Expected length=9, but was length=5.
Strings differ at index 0.

expected:<"bar - foo">
but was:<"zzzzz">
-----------^

at NUnit.Framework.Assert.Fail(String message, Object[] args)
at NUnit.Framework.Assert.AreEqual(Object expected, Object actual, String message, Object[] args)
at NUnit.Framework.Assert.AreEqual(Object expected, Object actual)
at Andy08 - nunit01Module.__Main__(String[] argv)
```

- though it's a bit verbose, and I got a windows dialog popping up asking if I want to debug the failure using visual studio etc. I said thanks but no thanks. Avoiding this pop up dialog would be good too.

Summary

Can anyone clarify the meaning of all the above findings? Can anyone show the recommended simplest way of building nunit tests for boo?

thanks,
-Andy

Didn't get an answer so I summarised thus:

Referring to assemblies with -r:
Date: Oct 29, 2004 - 3:16pm

If you refer to an assembly with either

```
booi.exe -r:SOMEASSEMBLY or
booc.exe -r:SOMEASSEMBLY
```

what happens when you refer to something that is not a "file dll on a path", but is rather a clean namespace reference. E.g. System.blah.blah - how does boo know where to find it? Is there a path similar to java's CLASSPATH that it/.NET consults?

And why would you want to refer to a System.blah.blah on the command line anyway (I saw this in a boo example of some doco somewhere), since all the System stuff should be accessible by default?

And why does booi.exe fail to find references whereas booc.exe succeeds? (see my earlier post on nunit).

thanks for any clarifications
Andy Bulka
http://www.atug.com/andypatterns

Got an answer :-)

From: Rafael Teixeira <monoman@gmail.com>
To: user@boo.codehaus.org, abulka@netspace.net.au
Date: Oct 30, 2004 - 12:31am

Inline
On Fri, 29 Oct 2004 15:16:59 +1000, Andy Bulka
<abulka@trinity.unimelb.edu.au> wrote:
> If you refer to an assembly with either
> booi.exe -r:SOMEASSEMBLY or
> booc.exe -r:SOMEASSEMBLY
> what happens when you refer to something that is not a "file dll on a path",
> but is rather a clean namespace reference.

You have to give assemblies names, not namespaces, as references,
ALWAYS. Some assemblies names coincide with the namespace for most of
the types it contains, but that is just a common design. Remember an
assembly may have types pertaining to lots of namespaces and the types
for a single namespace may reside each one in a different assembly.

>E.g. System.blah.blah - how does boo know where to find it?

It calls the System.Reflection.Assembly.Load() method. That means that
the system libraries decide how to find it. The main rule is try to
find it in the same directory where your program resides, or in the
GAC (Global Assembly Cache). The application can be configured to
search at another places (relative paths that must be rooted in the
application directory). The problem is the GAC is just for runtime, at
compile time you have to find the referenced assemblies somewhere
else.

>Is there a path similar to java's CLASSPATH that it/.NET consults?

Only in MONO (MONO_PATH environment var), but its use is not encouraged.

The best pratice is to use full paths to the dlls in the references.

> And why would you want to refer to a System.blah.blah on the command line
> anyway (I saw this in a boo example of some doco somewhere), since all the
> System stuff should be accessible by default?

Why should you pay the price to load all such big beasties always? And
what should always be loaded, there are dozens of "System...."
assemblies? The c# compiler in .NET has a default response file that
it uses to load default references. We could add such a thing to
booc/booi, but it's better for you to control it explicitly in the
makefile or in a response file specific to your project.

> And why does booi.exe fail to find references whereas booc.exe succeeds?
> (see my earlier post on nunit).

Well booc generates an executable assembly and so relative references
can be resolved in an easier fashion. Using fully qualified paths
solve all problems.

> thanks for any clarifications
> Andy Bulka
> http://www.atug.com/andypatterns

Hope it clarifies things a bit,

--
Rafael "Monoman" Teixeira

Just the 'crazy' me in a sane world, or would it be the reverse? I dunno...

Got a clarification

From: Rodrigo B. de Oliveira <rodrigobamboo@gmail.com>
To: user@boo.codehaus.org, abulka@netspace.net.au
Reply-To: Rodrigo B. de Oliveira <rodrigobamboo@gmail.com>
Date: Oct 30, 2004 - 4:42am

Hi Andy!

On Fri, 29 Oct 2004 15:16:59 +1000, Andy Bulka
<abulka@trinity.unimelb.edu.au> wrote:
> If you refer to an assembly with either
> booi.exe -r:SOMEASSEMBLY or
> booc.exe -r:SOMEASSEMBLY

Just clarifying one point: booi currently doesnt accept any arguments
but the script to be run. It sucks. we should make a move to
Mono.GetOptions and unify command line argument handling for both.
booc should be translated to boo in the process :)

> ...
> And why would you want to refer to a System.blah.blah on the command line
> anyway (I saw this in a boo example of some doco somewhere), since all the
> System stuff should be accessible by default?
>

Yeah. I don't know. I tend to use the import <namespace> from
<assembly> for quick throw away scripts. But when the code starts to
grow I tend to move to a more disciplined build/run cycle with all the
references specified externally.

> And why does booi.exe fail to find references whereas booc.exe succeeds?
> (see my earlier post on nunit).
>

booi.exe currently sucks.

cheers,
Rodrigo

Got a comment relating to Mono

From: Bet's On! <l33ts0n@gmail.com>
To: user@boo.codehaus.org, Rodrigo B. de Oliveira <rodrigobamboo@gmail.com>
Date: Oct 30, 2004 - 8:50am

Won't that tie Boo to Mono (On Windows,at least)?

reply 1:

From: Rodrigo B. de Oliveira <rodrigobamboo@gmail.com>
To: Bet's On! <l33ts0n@gmail.com>
CC: user@boo.codehaus.org
Date: Oct 30, 2004 - 8:55am

Mono.GetOptions does not depend on mono in any way.

And as far as I understand we could include it in our repository and
distros since it's licensed under X11.

But I'm still not sure about adding yet another assembly to our
growing collection of binaries. I was already thinking on including
antlr's source files to Boo.AntlrParser to ge rid of
antlr.runtime.dll. Maybe we could do the same with Mono.GetOptions...

On Fri, 29 Oct 2004 18:50:09 -0400, Bet's On! <l33ts0n@gmail.com> wrote:
> Won't that tie Boo to Mono (On Windows,at least)?
> ...

reply 2:

From: Rafael Teixeira <monoman@gmail.com>
To: user@boo.codehaus.org, Rodrigo B. de Oliveira <rodrigobamboo@gmail.com>
CC: Bet's On! <l33ts0n@gmail.com>
Date: Oct 30, 2004 - 1:43pm

On Fri, 29 Oct 2004 20:55:22 -0200, Rodrigo B. de Oliveira
<rodrigobamboo@gmail.com> wrote:
> Mono.GetOptions does not depend on mono in any way.

Precisely, it uses only corlib/System.

> And as far as I understand we could include it in our repository and
> distros since it's licensed under X11.

Yes, that was the idea.

> But I'm still not sure about adding yet another assembly to our
> growing collection of binaries. I was already thinking on including
> antlr's source files to Boo.AntlrParser to ge rid of
> antlr.runtime.dll. Maybe we could do the same with Mono.GetOptions...

Please do it if you really need, but I think doing a zip installer (a
self-extracting installer or runner) is better.
I intend to write one in the near future. Does someone care to help?

> On Fri, 29 Oct 2004 18:50:09 -0400, Bet's On! <l33ts0n@gmail.com> wrote:
> > Won't that tie Boo to Mono (On Windows,at least)?
> > ...
>

Current State of writing unit tests using Boo

I don't know what to make of all the above information. I would like a definitive how-to answer, rather than more experimenting and prodding at things I don't quite understand.

Contributions

Please add your contributions! I will summarise and make a wikki page on the Boo confluence site.

Date:
3/29/2005
Time:
8:00:51 PM
Remote User:

Comments

Has booi.exe been improved since the time of this set of posts? (about oct 2004).

Date:
3/29/2005
Time:
10:17:24 PM
Remote User:

Comments

I've combined booi, booc, and booish into one tool. It supports references, multiple files, etc., and uses Mono.GetOptions. It likely will be included in a future version of boo. http://jira.codehaus.org/browse/BOO-209 Have you tried the SharpDevelop addin? It is even easier to use for compiling boo scripts.

Date:
3/30/2005
Time:
2:57:50 AM
Remote User:

Comments

Date: Tues, Mar 29 2005 10:11 pm From: Bet's On!

(posted to list for your and everyone else's benefit)

Here's your problem:

1: booi.exe is an interperter and does not support "-r." If you installed NUnit, you don't need the reference; it will be installed to the GAC, so all you need is "import NUnit.Framework"
2: You can compile
3 types of assemblies in Boo, just like in C#. Libraries, Console Applications, and Windows Applications.

By default, booc.exe will try to compile a Console Application. This means that your source file needs an "entry point"; basically, code that is not encapsulated into a class. To avoid this, visit the Boo wiki about "how to compile" and use the command-line switch (the switch escapes me at the moment) to compile to a library.

You get popups because you're not supposed to "run" Unit Tests; they're supposed to be run from the command-line using Nunit or the user interface for Nunit. Boo has a handy-dandy built-in "assert" keyword that is nowhere near as verbose as NUnit's Assert class.