

How To Compile

There are basically three ways to compile your boo code:

- through the **booc** utility;
- through the **booc** task (one of the [nant tasks](#));
- through the Boo.Lang.Compiler API;

booc utility

```
booc -r:System.Windows.Forms
-out:build/HelloForms.exe
examples/helloforms.boo
```

```
booc -t:library -out:build/MyLibrary.dll
alibrary.boo
```

Command line arguments

- -v, -vv and -vvv - Compiles using verbose mode
- -r:<reference_name> - adds a reference, where <reference_name> is the library's name. For example:

```
booc -r:Utilities.dll MyProgram.boo
```

- -o:<file_name_to_generate>: Forces the generated file to be the specified by <file_name_to_generate>. So,

```
booc -o:SpongeBob.exe MyProgram.boo
```

will create an .exe file named *SpongeBob.exe* instead of *MyProgram.exe*

- -t:<type_name_to_generate> - Forces the generation of <type_name_to_generate>. Options may be *library* - to create a .dll file -, *exe* and *winexe* to create executables.
- -p:<pipeline> - adds the step <pipeline> to the compile process
- -c:<culture> - *CultureInfo* to use
- -srcdir:<source_files> - specify where to find the source files.
- -resource:<file.resources> - specify a resource file

booc task

```
<booc target="winexe"  
output="build/HelloForms.dll">  
  <sources basedir="examples">  
    <include name="HelloForms.boo" />  
  </sources>  
  
  <references>  
    <include  
name="System.Windows.Forms.dll" />  
  </references>  
</booc>
```

Look [here](#) for details.

Boo.Lang.Compiler API

```
import Boo.Lang.Compiler
import Boo.Lang.Compiler.IO
import Boo.Lang.Compiler.Pipelines

// create a compiler object
compiler = BooCompiler()

// set the inputs
compiler.Parameters.Input.Add(StringInput("<
script>", "print('Hello!')"))

// set the output
compiler.Parameters.OutputAssembly =
"test.exe"

// configure the pipeline
compiler.Parameters.Pipeline =
CompileToFile()

// run
result = compiler.Run()

// check for errors
print(join(result.Errors)) if
len(result.Errors)
```

Running & Compiling boo programs that use Mono or GTK# on Windows

Here are instructions for running and compiling boo scripts that use Mono-only assemblies (such as Mono.GetOptions.dll or gtk-sharp.dll) on a Windows machine. I'm assuming you are already familiar with using the Windows command prompt (the [open command here XP powertoy](#) is helpful).

- [Download](#) the Windows installer for Mono that includes GTK# and XSP and run it.
- You might want to add mono to your Path environment variable. Right click My Computer, go to properties, advanced, environment variables. Edit Path by appending a semicolon (;) followed by "C:\Program Files\Mono-1.0.4\bin" (no quotes).
- You can test two scripts in the examples directory to see if it is working:

```
mono bin\booc.exe -r:gtk-sharp -out:bin\gtk.exe examples\gtk.boo  
  
mono bin\booc.exe -r:Mono.GetOptions -out:bin\GetOptions.exe  
examples\GetOptions.boo
```

- Then run the applications to test them. They should run cross-platform.

```
mono bin\GetOptions.exe
```

- If you want the gtk-sharp or mono.getoptions dlls for redistributing your app (although see the previous update note: now you can't run a mono app in .NET), you will find the dlls you need in C:\Program Files\Mono-1.0.2\lib\mono and C:\Program Files\Mono-1.0.2\lib\mono\1.0.
- You can now develop completely cross-platform applications even if you don't have access to a Linux machine.

Distributing Your Boo App or Library, Merging Boo.Lang.dll

Pretty much any application or dll library you compile in boo will depend on the Boo.Lang.dll. You need to include that dll with your application or library when distributing it.

You can use a tool called ILMerge to combine the Boo.Lang.dll into your exe or dll. See [Merge Boo.Lang.dll into your exe or dll](#).