Part 19 - Using the Boo Compiler

Part 19 - Using the Boo Compiler

The Boo Compiler is typically called in this fashion:
`booc <options> <files>`

Command-line Options

Option	Description
`-v`	Verbose
`-vv`	More Verbose
`-vvv`	Most Verbose
`-r:`*<reference_name>*	Add a reference to your project
`-t:`*<type_name_to_generate>*	Type of file to generate, can be either `exe` or `winexe` to make executables (.exe files), or `library` to make a .dll file
`-p:`*<pipeline>*	Adds a step *<pipeline>* to the compile.
`-c:`*<culture>*	Sets which *CultureInfo* to use.
`-o:`*<output_file>*	Sets the name of the output file
`-srcdir:`*<source_files>*	Specify where to find the source files.
`-debug`	Adds debug flags to your code. Good for non-production. (On by default)
`-debug-`	Does not add debug flags to your code. Good for production environment.
`-debug-steps`	See AST after each compilation step.
`-resource:`*<resource_file>*`,`*<name>*	Add a resource file. *<name>* is optional.
`-embedres:`*<resource_file>*`,`*<name>*	Add an embedded resource file. *<name>* is optional.

So, for example, in order to compile your Database code that depends on the library System.Data.dll, you would type: `booc -r:System.Data.dll -o:Database.dll -t:library Database.boo`

That would create a fully functional, working compilation of your library: `Database.dll`

Using NAnt

When working on a large project with multiple files or libraries, it is a lot easier to use NAnt. It is a free .NET build tool.

To do the same command as above, you would create the following build file:

default.build

```
<?xml version="1.0" ?>

<project name="Goomba" default="build">
 <target name="build" depends="database" />
      <target name="database">
                <mkdir dir="bin" />
              <booc output="bin/Database.dll" target="library">
                        <references basedir="bin">
                               <include name="System.Data.dll" />
                      </references>
                   <sources>
                                <include name="Database.boo" />
                 </sources>
               </booc>
 </target>
</project>
```

```
$ nant
NAnt 0.85 (Build 0.85.1869.0; rc2; 2/12/2005)
Copyright (C) 2001-2005 Gerry Shaw
http://nant.sourceforge.net

Buildfile: file:///path/to/default.build
Target framework: Microsoft .NET Framework 1.1
Target(s) specified: build

build:

database:

     [booc] Compiling 1 file(s) to /path/to/bin/Database.dll.

BUILD SUCCEEDED

Total time: 0.2 seconds.
```

And although that was a long and drawnout version of something so simple, it does make things *a lot* easier when dealing with multiple files.

It also helps that if you make a change to your source files, you don't have to type a long `booc` phrase over again.

The important part of the build file is the `<booc>` section. It relays commands to the compiler.
There are four attributes available to use in it:

Attribute	Description
`target`	Output type, one of `library`, `exe`, `winexe`. Optional. Default: `exe`.
`output`	The name of the output assembly. **Required**.
`pipeline`	AssemblyQualifiedName for the CompilerPipeline type to use. Optional.
`tracelevel`	Enables compiler tracing, useful for debugging the compiler, one of: Off, Error, Warning, Info, Verbose. Optional. Default: Off.

You are most likely only to use `target` and `output`.

For nested elements, you have 3 possibilities:

Nested Element	Description
`<sources>`	Source files. **Required**.
`<references>`	Assembly references.
`<resources>`	Embedded resources.

Inside these you are to put `<include />` elements, as in the example.

This is merely a brief overview of NAnt, please go to their website http://nant.sourceforge.net for more information.

Go on to **Part 20 - Structure of a Boo Project**