Continuous Integration with CruiseControl.Net

Part 1

Paul Grenyer

What is Continuous Integration?

Continuous integration is vital to the software development process if you have multiple build configurations and/or multiple people working on the same code base. WikiPedia [WikiPedia] describes continuous integration as "…a software engineering term describing a process that completely rebuilds and tests an application frequently." Generally speaking continuous integration is the act of automatically building a project or projects and running associated tests; typically after a checkin or multiple checkins to a source control system.

Why Should You Use Continuous Integration?

My Dad takes a huge interest in my career and always wants to know what I'm doing. He knows almost nothing about software engineering, but he does know *a lot* about cars, so I often use the production of a car as an analogy to software engineering.

Imagine a car production factory where one department designs and builds the engine, another department designs and builds the gear box and a third department designs and builds the transmission (prop shaft, differential, etc). The engine has to connect to the gearbox and the gearbox to the transmission. These departments work to a greater or lesser degree in isolation, just like teams or individuals working on different libraries or areas of a common code base on some projects.

A deadline has been set in the factory for all the parts of the car to be ready and the car will be assembled and shipped the next day. During the time to the deadline the gearbox is modified to have four engine mountings, as a flaw in the original design is identified, instead of the three the specification dictates and the ratio of the differential is changed as the sales department has promised the customer the car will have a higher top speed.

The deadline has arrived and the first attempt to assemble the engine, gearbox and transmission is made. The first problem is that the gearbox cannot be bolted onto the engine correctly as there are insufficient mountings on the engine. However this can be fixed, but will take an extra two weeks while the engine block is recast and the necessary mountings added.

Two weeks later the engine, gearbox and transmission are all assembled, bolted into the car and it's out on the test track. The car is flat out down the straight and it is 10 miles per hour slower than sales department promised it would be as the engine designers did not know about the change in differential ratio and the maximum torque occurs at the wrong number of revs. So the car goes back to the factory have the valve timings adjusted which takes another two weeks.

When presented like this it is clear that there is a problem with the way development has been managed. But all too often this the way that software development is done - specs are written and software developed only to be put together under the pressure of the final deadline. Not surprisingly the software is delivered late, over budget and spoils reputations. We've all been there.

The problems could have been avoided or at least identified in time to be addressed, by scheduling regular integrations between the commencement of production and the deadline. Exactly the same applies to software engineering. All elements of the system should be built together and tested regularly to make sure that it builds and that it performs as expected. The ideal time is every time a checkin is made. Integration problems are then picked up as soon as they are created and not the day before the release and the ideal way to do this is using an automated system such as CruiseControl.

CruiseControl.Net

CruiseControl, written in Java, is one of the better known continuous integration systems it is designed to monitor a source control system, wait for checkins, do builds and run tests. CruiseControl.Net [CruiseControl.Net] is, obviously, a .Net implementation of CruiseControl, designed to run on Windows although it can be used with Mono[Mono].

I found the simple start-up documentation for CruiseControl.Net sadly lacking, so in this article I am going to go through a simple CruiseControl.Net configuration step-by-step using my Aeryn [Aeryn] C++ testing framework. Aeryn is an ideal example as it has both Makefiles for building in Unix-like environments and a set of Microsoft Visual C++ build files. It also has a complete test suite which is run as part of the build.

Download and Install

You can download CruiseControl.Net from the CruiseControl.Net website. It comes in several different formats including source and a Windows MSI installer. Download and install the Windows MSI and select the defaults. This will install CruiseControl.Net as a service and setup a virtual directory that so it can be used with Microsoft's Internet Information Service [IIS] to give detailed information about the builds (I'll cover this in Part 2). Also download and install the CCTray Windows MSI. CCTray is a handy utility for monitoring builds I'll discuss later.

CruiseControl.Net can run as both a command line program and a Windows service. It is useful to start off with the command line version and then move to the Windows service once all the configuration bugs have been ironed out.

Project Block

CruiseControl.Net uses an XML configuration file called `ccnet.config`, which is located in the CruiseControl.Net `server` directory (the default CruiseControl.Net install directory is: `C:\Program Files\CruiseControl.NET`). The configuration must be wrapped in a `<cruisecontrol>` block and contain at least one `<project>` block:

```
<cruisecontrol>
       <project name="Aeryn" >
       </project>
</cruisecontrol>
```

The above is the minimal project block. In the above example the project is simply given the name Aeryn. It will be added as we step through the configuration. To run CruiseControl.Net from the command line, open a command prompt and change to the `server` directory, type `ccnet` and hit return:

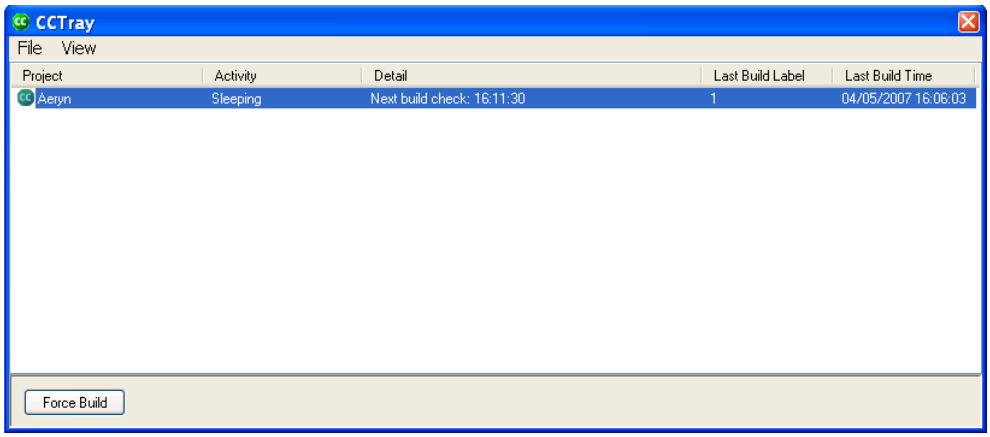
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>cd C:\Program Files\CruiseControl.NET\server

C:\Program Files\CruiseControl.NET\server>ccnet
CruiseControl.NET Server 1.2.1.7 -- .NET Continuous Integration Server
Copyright (C) 2003-2006 ThoughtWorks Inc.  All Rights Reserved.
.NET Runtime Version: 2.0.50727.42      Image Runtime Version: v1.1.4322
OS Version: Microsoft Windows NT 5.1.2600 Service Pack 2        Server locale: en-
GB

[CCNet Server:DEBUG] The trace level is currently set to debug.  This will cause
CCNet to log at the most verbose level, which is useful for setting up or
debugging the server.  Once your server is running smoothly, we recommend changing
this setting in C:\Program Files\CruiseControl.NET\server\ccnet.exe.config to a
lower level.
[CCNet Server:INFO] Reading configuration file "C:\Program
Files\CruiseControl.NET\server\ccnet.config"
[CCNet Server:INFO] Registered channel: tcp
[CCNet Server:INFO] CruiseManager: Listening on url:
tcp://192.168.0.100:21234/CruiseManager.rem
[CCNet Server:INFO] Starting CruiseControl.NET Server
[Aeryn:INFO] Starting integrator for project: Aeryn
[Aeryn:INFO] No modifications detected.
```

This starts CruiseControl.Net, but it has nothing to do so it just sits and waits. Now is a good point at which to configure CCTray. Bring up the CCTray window by double clicking on the CCTray icon (usually a green, red or orange circle with CC in the centre) in the system tray. First register the CruiseControl.Net server:

1. Select the file menu and settings.
2. Then select the Add button form the Build Projects tab.
3. Click the Add Server button from the project dialog.
4. Select the “Connect directly using .Net remoting” radio button.
5. Enter `localhost` to connect to a server on the local machine or the IP address or host name for a server on a remote machine and click Ok.
6. Select the project (in this case Aeryn) from the projects list box and click Ok.

7. Click Ok on the CruiseControl.Net Tray Settings dialog.

CCTray will connect to the CruiseControl.Net server and you should see something like this:



CCTray Main Window

CCTray is designed not only to run on the same machine as CruiseControl.Net, but on any number of client machines as well.

Source Control Block

CruiseControl.Net can be configured to monitor a number of source control systems for changes. Theses include subversion, CVS, Perforce, ClearCase and Visual Source Safe. The CruiseControl.Net documentation includes a complete list. One of the strengths of CruiseControl.Net is that it is easy to add support, via a plugin, for other source control systems. I plan to write about creating CruiseControl.Net plugins in future articles.

Aeryn uses Subversion [SVN] and CruiseControl.Net. A subversion client must be installed to use it (TortoiseSVN doesn't appear to have the right executable). The minimum parameters needed are the (trunk) URL of the repository and the working directory (a path to check the code out to). However, this assumes that `svn.exe` (subversion client executable) is also in the working directory, so it is necessary to specify the path to it. The full working directory must also exist.

```
<project name="Aeryn">
	<sourcecontrol type="svn">
		<trunkUrl>http://aeryn.tigris.org/svn/aeryn/trunk/</trunkUrl>

		<workingDirectory>c:\temp\ccnet\aeryn</workingDirectory>
		<executable>C:\Program Files\Subversion\bin\svn.exe</executable>
	</sourcecontrol>
</project>
```

CruiseControl.Net monitors `ccnet.config`, so simply making the above changes and saving the file should be all that needs to be done. Alternatively the server can be started again from the command line.

```
…
[CCNet Server:DEBUG] The trace level is currently set to debug.  This will cause CCNet to log at the most verbose level,
 which is useful for setting up or debugging the server.  Once your server is running smoothly, we recommend changing th
is setting in C:\Program Files\CruiseControl.NET\server\ccnet.exe.config to a lower level.
[CCNet Server:INFO] Reading configuration file "C:\Program Files\CruiseControl.NET\server\ccnet.config"
[CCNet Server:INFO] Registered channel: tcp
[CCNet Server:INFO] CruiseManager: Listening on url: tcp://192.168.0.100:21234/CruiseManager.rem
[CCNet Server:INFO] Starting CruiseControl.NET Server
[Aeryn:INFO] Starting integrator for project: Aeryn
...
[Aeryn:INFO] No modifications detected.
```

There is some extra debug information, such as the last checkin message, omitted from the above output. The final message is `No modifications detected`. This means that CruiseControl.Net has identified that there have been no recent changes to the repository. Therefore it has not checked anything out and it has not attempted to build anything. One way to test that it checks code out correctly would be to commit a change to the repository, however this is unnecessary. CCTray can be used to force the checkout. Select the project from the CCTray list box and click Force Build.

```
[Aeryn:DEBUG] Starting process [C:\Program Files\Subversion\bin\svn.exe] in
working directory [c:\temp\ccnet\aeryn] with arguments [log
http://aeryn.tigris.org/svn/aeryn/trunk/ -r "{2007-05-04T16:06:42Z}:{2007-05-
04T17:26:17Z}" --verbose --xml --non-interactive --no-auth-cache]
...
[Aeryn:INFO] No modifications detected.
[Aeryn:INFO] Building: Paul Grenyer triggered a build (ForceBuild)
[Aeryn:DEBUG] Starting process [C:\Program Files\Subversion\bin\svn.exe] in
working directory [c:\temp\ccnet\aeryn] with arguments [checkout
http://aeryn.tigris.org/svn/aeryn/trunk/ c:\temp\ccnet\aeryn --non-interactive --
no-auth-cache]
[Aeryn:DEBUG] A    C:\temp\ccnet\aeryn\corelib
[Aeryn:DEBUG] A    C:\temp\ccnet\aeryn\corelib\corelib.vcproj
[Aeryn:DEBUG] A    C:\temp\ccnet\aeryn\corelib\Makefile
[Aeryn:DEBUG] A    C:\temp\ccnet\aeryn\Doxyfile
[Aeryn:DEBUG] A    C:\temp\ccnet\aeryn\include
...
[Aeryn:DEBUG] A    C:\temp\ccnet\aeryn\examples\lift\TestClient\main.cpp
[Aeryn:DEBUG] U   C:\temp\ccnet\aeryn
[Aeryn:DEBUG] Checked out revision 157.
[Aeryn:INFO] Integration complete: Success - 04/05/2007 18:26:50
```

The SVN source control block, unlike some of the other source control blocks, supports username and password parameters. This allows code to be checked out on machines where the current user, such as the system account if CruiseControl.Net is running as a service, does not have the necessary permissions.

```
<sourcecontrol type="svn">
        <trunkUrl>http://aeryn.tigris.org/svn/aeryn/trunk/</trunkUrl>
        <workingDirectory>c:\temp\ccnet\aeryn</workingDirectory>
        <executable>C:\Program Files\Subversion\bin\svn.exe</executable>
        <username>fprefect<username>
        <password>towel<password>
</sourcecontrol>
```

The drawback is that the username and password are stored in `ccnet.config` in unencrypted human readable format. However, CruiseControl.Net only needs to be able to check code out, it doesn't need to check it back in, so if the repository you are using supports anonymous checkouts this is less of a disadvantage.

Devenv Task Block (Visual Studio 7.x Task)

One of the two basic build systems supported by Aeryn is Microsoft Visual Studio solutions. CruiseControl.Net has two special task blocks for visual studio solutions: `<devenv>` and `<msbuild>`. `<devenv>` is used to build version 7.x solutions and `<msbuild>` to build version 8 solutions using Microsoft's MSBuild [MSBuild]. Aeryn uses visual studio 7.1 solutions and therefore requires `<devenv>`.

The minimum parameters needed are the solution file to build and the configuration to build (e.g. `debug` or `release`). However this assumes that Visual Studio 7.x is installed at a specific location, but Visual Studio 7.x can be installed to any path and the default path varies across versions, so it is best to specify the path to the `devenv.com` executable for the version being used.

```
<sourcecontrol type="svn">
        ...
</sourcecontrol>
<tasks>
        <devenv>
                <solutionfile>C:\temp\ccnet\aeryn\aeryn2.sln</solutionfile>
                <configuration>Debug</configuration>
                <executable>C:\Program Files\Microsoft Visual Studio .NET…
                                        …2003\Common7\IDE\devenv.com</executable>
        </devenv>
</tasks>
```

There are a number of other useful `<devenv>` parameters, two of which are: `<buildtype>` and `<buildTimeoutSeconds>`. The build types are `Build`, `Clean` and `Rebuild` and have their normal visual studio meanings. The default is `Rebuild`. Build timeout is the number of seconds that CruiseControl.Net will wait before assuming the build has hung and should be killed. The default is 600 (10mins):

```
<devenv>
        <solutionfile>C:\temp\ccnet\aeryn\aeryn2.sln</solutionfile>
        <configuration>Debug</configuration>
        <executable>C:\Program Files\Microsoft Visual Studio .NET…
                                        …2003\Common7\IDE\devenv.com</executable>
        <buildtype>Rebuild</buildtype>
        <buildTimeoutSeconds>300</buildTimeoutSeconds>
</devenv>
```

Aeryn has both a debug and a release configuration and the building of both should be tested. That requires two `<devenv>` blocks and two fully hard coded solution paths. This introduces a possible maintenance headache if the working directory is moved. Solution paths can be relative if a working directory is specified in the project block.

```
<cruisecontrol>
	<project name="Aeryn">
		<workingDirectory>c:\temp\ccnet\aeryn</workingDirectory>
		...
		<tasks>
			<devenv>
				<solutionfile>aeryn2.sln</solutionfile>
				<configuration>Debug</configuration>
				<executable>C:\Program Files\Microsoft Visual…
			…Studio .NET 2003\Common7\IDE\devenv.com</executable>
				<buildtype>Rebuild</buildtype>
				<buildTimeoutSeconds>300</buildTimeoutSeconds>
			</devenv>
			<devenv>
				<solutionfile>aeryn2.sln</solutionfile>
				<configuration>Release</configuration>
				<executable>C:\Program Files\Microsoft Visual…
			…Studio .NET 2003\Common7\IDE\devenv.com</executable>
				<buildtype>Rebuild</buildtype>
				<buildTimeoutSeconds>300</buildTimeoutSeconds>
			</devenv>
		</tasks>
		...
	</project>
</cruisecontrol>
```

Again, the changes to `ccnet.config` should be automatically picked up by the server when it is saved or the server can be restarted from the command line. Using CCTray to force the build will cause both configurations to build.

Exec (make) Task Block

The other build system supported by Aeryn is make on both Windows and Linux. Obviously CruiseControl.Net can only run the Windows version. CruiseControl.Net doesn't have a specific make task block, so a generic executable block must be used instead.

I think that a task block supporting make is a fundamental omission from CruiseControl.Net. As creating new task blocks is very easy; I have written a make task block and am currently trying to get it incorporated into CruiseControl.Net. If I am unsuccessful I will be making it available as a plugin.

The parameters are the path to the executable, the arguments to pass to the executable, the number of seconds to wait before assuming that the process has hung and should be killed, and the working directory. The working directory is only needed if it is different from the working directory specified by the project block.

```
<exec>
	<executable>C:\MinGW\bin\mingw32-make.exe"</executable>
	<buildArgs>-f Makefile rebuild</buildArgs>
	<buildTimeoutSeconds>300</buildTimeoutSeconds>
</exec>
```

This disadvantage of using an `<exec>` block to run make is that it does not give any `INFO` level log output indicating that the task is running or whether it was successful, as the `<devenv>` and `<msbuild>` blocks do. CruiseControl.Net also generates a `DEBUG` level error as the output from calling make is not in XML format. This is not a serious problem as when CruiseControl.Net is running in production debug logging should be turned off. The make task block I have written solves both these issues.

Saving the changes to `ccnet.config` and using CCTray to force the build should cause the make configurations to build along with the visual studio configurations.

Publisher Block - Email

CruiseControl.Net uses publisher blocks to notify developers of the status of the build. The most useful publisher block is email. The email block can be used to send status emails to groups of email addresses.

For example the developer responsible for maintaining the CruiseControl.Net server may want an email every time a build takes place. However, the rest of the developers on the team probably only want to receive an email when the status of the build changes (e.g. from fixed to broken or vice-versa) or while the build is broken. To achieve this, two groups can be setup, a “developers” group and a “buildmaster” group, each group is configured individually.

Emails can be triggered by three different notification events:

Always	An email is sent every time a build takes a place.
Change	An email is sent when the status of the build changes. Either from fixed to broken or broken to fixed.
Failed	Sends an email whenever a build fails.

An SMTP server must also be specified along with the relevant username and password when needed. The `<email>` block has the same security issues as the source control block in terms of the username and password being in human readable format. However, most of the time a build server will be fixed within a particular network and access to the SMTP server on a corporate network or through the ISP will not require a username or password.

If CruiseControl.Net is being run on a roaming computer such as a laptop then this becomes more of an issue. I use Google mail for my every day email and the Google SMTP server uses a non-standard port and requires a secure connection. This is not supported by the email block or, it appears, the underlying .Net SMTP class. I am sure that both the username and password issue and the port and security issue could be overcome by writing a custom email block based on the existing one, however that is outside the scope of this article.

```
<project name="Aeryn">
	...
	<publishers>
		<email from=paul.grenyer@gmail.com...
		 …mailhost="mailhost.zen.co.uk" includeDetails="TRUE">
		<users>
			<user name="Paul Grenyer" group="buildmaster"…
				…address="paul.grenyer@gmail.com"/>
			<user name="Aeryn Developers" group="developers"…
			…address="continuousintegration@aeryn.tigris.org"/>
		</users>
		<groups>
			<group name="developers" notification="change"/>
			<group name="buildmaster" notification="always"/>
		</groups>
		</email>
	</publishers>
</project>
```

Assuming the SMTP details and email addresses are correct, emails will be sent at the end of each build.

When setting up email notifications from continuous integration on multi-developer projects it is important to be aware of how the members of the team feel about potentially receiving a lot of extra email and having the fact that their code changes have broken the build highlighted to the team.

During the setting up of the server it is sensible to restrict emails to the person doing the setup. I found that people became irritated with only a small increase in email to begin with. However as the builds became more successful and the appropriate email rules implemented (see above), this became less of an issue.

To get people to accept that they have broken the build and agree to fix it, I found that it was important to get buy-in for continuous integration. This is an ongoing task. The management, however, is on side and pushing quite hard. I am sure that as soon as the next release is built smoothly everyone will be more enthusiastic about continuous integration and maintaining working builds.

Running CruistControl.Net as a Service

CruiseControl.Net can be run as a Windows Service. This has the advantage that whenever the dedicated build server is rebooted or someone logs in or logs out, the CruiseControl.Net server keeps running. Running CruiseControl.Net as a service uses exactly the same `ccnet.config` file.

During the development of the configuration it useful to have lots of debug information. Once the configuration is complete and working; this extra debug information is no longer useful and should be turned off. To adjust the logging level edit the `<level value>` tag in the `ccservice.exe.config` file in the CruiseControl.Net `server` directory. The available levels are `DEBUG`, `INFO`, `WARN`, `ERROR`, `OFF`. The default is `Debug`. Changing the setting to `INFO` reduces a lot of unnecessary noise.

```
<level value="INFO" />
```

Changes to `ccservice.exe.config` are not picked up by CruiseControl.Net until it is restarted.

CruiseControl.Net is installed as a Windows service as part of the standard setup. Before it can be started it must be configured to run as a user that has access to the source control system (unless the username and password have been put into source control block) and the compilers and applications that are used in the configuration.

1. Open the Services dialog (Control Panel->Administrative Tools->Services) and double click on CruiseControl.Net Server.
2. In the General Tab set Start Type to Automatic so that CruiseControl.Net starts when the build server starts.
3. In the Log On tab select the This Account radio button and enter the username and password of a user who has rights to the necessary source control and application.
4. Click Ok and use the Services dialog to start CruiseControl.Net.

CruiseControl.Net writes a log file called `ccnet.log` to the `Service` directory. It can be useful to monitor this with a tool such as tail [Tail]. When CruiseControl.Net is running as a service a build can be forced from CCTray in the same way as when it was running from the command line.

Final Test

As a final test modify a source file in such a way as to break the build. Commit the file to the source control system and see that it triggers the CruiseControl.Net build and that the build fails. Then undo the modification, commit it again and see that the build succeeds.

`ccnet.config` should also be committed to the source control system and checking it in will also trigger a build.

Part 2

In this article I have demonstrated how easy it is to setup continuous integration with CruiseControl.Net. I found the CruiseControl.Net documentation lacking and I hope this article has started to rectify that situation. I have highlighted some of CruiseControl.Net's shortcomings, not only it's lack of documentation, but the fact it is missing at least one fundamentally important task block and that the source control and email blocks need to have additional features. I intend to address these issues, with plugins in future articles.

I think it is clear from the fact that continuous integration highlights integration issues early that it is something we should all be doing on multi-developer projects or projects with multiple build systems. This is certainly what I have found in the relatively short period of time I have been using continuous integration.

In part two of Continuous Integration with CruiseControl.net I am going to look at setting up a webserver to allow more detailed information to be obtained about the status of the CruiseControl.Net server and its projects and builds.

Complete `ccnet.config` File

```
<cruisecontrol>
	<project name="Aeryn">
		<workingDirectory>c:\temp\ccnet\aeryn</workingDirectory>
		<sourcecontrol type="svn">
			<trunkUrl>http://aeryn.tigris.org/svn/aeryn/trunk/</trunkUrl>
			<workingDirectory>c:\temp\ccnet\aeryn</workingDirectory>
			<executable>C:\Program Files\Subversion\bin\svn.exe</executable>
		</sourcecontrol>
		<tasks>
			<devenv>
				<solutionfile>aeryn2.sln</solutionfile>
				<configuration>Debug</configuration>
				<executable>C:\Program Files\Microsoft Visual Studio…
					….NET 2003\Common7\IDE\devenv.com</executable>
				<buildtype>Rebuild</buildtype>
				<buildTimeoutSeconds>300</buildTimeoutSeconds>
			</devenv>
			<devenv>
				<solutionfile>aeryn2.sln</solutionfile>
				<configuration>Release</configuration>
				<executable>C:\Program Files\Microsoft Visual Studio…
					….NET 2003\Common7\IDE\devenv.com</executable>
				<buildtype>Rebuild</buildtype>
				<buildTimeoutSeconds>300</buildTimeoutSeconds>
			</devenv>
			<exec>
				<executable>C:\MinGW\bin\mingw32-make.exe</executable>
				<buildArgs>-f Makefile rebuild</buildArgs>
				<buildTimeoutSeconds>300</buildTimeoutSeconds>
			</exec>
		</tasks>
		<publishers>
			<email from="paul.grenyer@gmail.com"...
				…mailhost="mailhost.zen.co.uk" includeDetails="TRUE">
				<users>
					<user name="Paul Grenyer" group="buildmaster"…
							…address="paul.grenyer@gmail.com"/>
					<user name="Aeryn Developers" group="developers"…
					…address="continuousintegration@aeryn.tigris.org"/>
				</users>
				<groups>
					<group name="developers" notification="change"/>
					<group name="buildmaster" notification="always"/>
				</groups>
			</email>
		</publishers>
	</project>
</cruisecontrol>
```

Acknowledgments

Thank you to Jez Higgins, Peter Hammond, Roger Orr, Paul Thomas and Alan Griffiths for reviews and suggestions.

References

[WikiPedia] http://en.wikipedia.org/wiki/Continuous_Integration
[CruiseControl.Net] http://ccnet.thoughtworks.com/
[Aeryn] http://www.aeryn.co.uk
[Mono] http://www.mono-project.com
[IIS] http://www.microsoft.com/windowsserver2003/iis/default.mspx
[SVN] http://subversion.tigris.org/
[MSBuild] http://msdn2.microsoft.com/en-us/library/wea2sca5(VS.80).aspx
[Tail] http://tailforwin32.sourceforge.net/