## Install And Build CruiseControl

First, download CruiseControl. Unzip this file to your applications directory. The top-level directory created when you unzipped this file (cruisecontrol-src-2.8.4) will be referred to as INSTALL_DIR.

There are several directories under INSTALL_DIR. We are primarily concerned with:

| Directory | Contents |
|---|---|
| contrib | Scripts and utilities contributed to CruiseControl, but not part of the binary distribution, including the distributed builder. |
| main | The build loop, which is a Java application that provides the core build scheduling functionality of CruiseControl. |
| reporting/jsp | The JSP reporting application, which is a J2EE web application for showing build results |
| reporting/dashboard | The dashboard application, a newer J2EE web application for showing build results |

To build the CruiseControl jars you will need to execute the Ant build scripts in INSTALL_DIR/main, INSTALL_DIR/reporting/jsp and INSTALL_DIR/reporting/dashboard. These build scripts not only compile the code but also run the unit tests and then package the output. Occasionally people encounter environment specific failures when the unit tests run (such as not having java on the executable path). While these failures should be reported to the mailing list so that the tests can be fixed typically the failing test can be safely skipped to allow the build to complete. You can skip the tests by specifying the test.skip property when running ant: ant -Dtest.skip=true

After building, to confirm that the installation is consistent, start a command shell and run the following command (make sure you replace INSTALL_DIR with the actual installation directory):

```
java -jar INSTALL_DIR/main/dist/cruisecontrol-launcher.jar
```

You should get a "No config file" message, along with usage details, and a cruisecontrol.log file will be created in the current directory. This means that CruiseControl is correctly installed and ready to run (given a valid configuration file).

The remaining examples require INSTALL_DIR/main/bin to be added to the system path.

## Running the Build Loop

### *Setup a Working Area*

Now you need to set up your CruiseControl working directories, and create a configuration file. These should be placed in a separate directory outside of INSTALL_DIR to simplify later CruiseControl upgrades.

As an example, you might create a working directory of /work/cc. This will be referred to as WORK_DIR for the remainder of this guide.

Create three subdirectories underneath WORK_DIR:

| Directory | Contents |
|---|---|
| projects | This is where CruiseControl checks out your project from version control. |
| logs | This is where CruiseControl will write its build reports. |
| artifacts | This is where CruiseControl can put any build output files that need to be kept. |

This directory structure is very common and works quite well, but it is possible to have a very different directory structure than the one described here. The paths to these directories are all specified in the configuration files and can be in arbitrary locations.

### *Checkout a Project*

Manually checkout the module for the project you want to build into WORK_DIR/projects. We will refer to your project name as MY_PROJECT_1, and assume that the CVS module has the same name. There will now be a directory called WORK_DIR/projects/MY_PROJECT_1.

### *Create a Delegating Build Script*

Create a new Ant script called build-MY_PROJECT_1.xml in WORK_DIR as follows:

```
<!-- Delegating build script, used by cruisecontrol to build MY_PROJECT_1.
    Note that the basedir is set to the checked out project -->
<project name="build-MY_PROJECT_1"
      default="build"
      basedir="projects/MY_PROJECT_1">
```

```
  <target name="build">
    <!-- Get the latest from CVS -->
    <cvs command="up -d -P"/>
    <!-- Call the target that does everything -->
    <ant antfile="build.xml" target="build-everything"/>
  </target>
</project>
```

This is a delegating build script, which CruiseControl will run to build your project. Basically, this build script should just call through to your project build file WORK_DIR/projects/MY_PROJECT_1/build.xml, as well as performing extra steps which are specific to the CruiseControl build, such as:

- getting the latest sources from version control
- tagging the version control tree after a successful build

Of course, if your build already does this, you won't need to add these commands to the delegating build script. Instead you could bypass the delegating build script completely and configure CruiseControl to invoke the project build script directly.

CruiseControl provides a property to the script named "label" that is incremented after successful builds. You can reference it in your ant script as you do any other property: ${label}. For an overview of all properties passed to your build scrip.

### *Configure the Build Loop*

By default, CruiseControl looks for a configuration file named config.xml in the current directory. Create the file WORK_DIR/config.xml as follows:

```
<cruisecontrol>
</cruisecontrol>
```

Try running CruiseControl from within WORK_DIR. That is, start a command shell, cd to WORK_DIR, type the following command and press enter:

Windows:

```
INSTALL_DIR/main/bin/cruisecontrol.bat
```

Unix:

```
INSTALL_DIR/main/bin/cruisecontrol.sh
```

(If you're using the binary release the equivalent files are in INSTALL_DIR.)

CruiseControl should start up correctly, stating "BuildQueue started". However, to get CruiseControl to actually do anything, you'll need to configure a project.

It is possible to run CruiseControl from another location, and specify the path to config.xml using the -configfile command-line option. Be warned: the paths in your config file are relative to the current directory, and not relative to the config file itself.

Modify config.xml to look like the following. The meaning of each of the elements is explained below.

```
<cruisecontrol>
 <project name="MY_PROJECT_1" buildafterfailed="true"[1]>
  <listeners>
   <currentbuildstatuslistener[2]
     file="logs/MY_PROJECT_1/status.txt"/>
  </listeners>

  <!-- Bootstrappers are run every time the build runs,
    *before* the modification checks -->
  <bootstrappers>
  </bootstrappers>

  <!-- Defines where CruiseControl looks for changes, to decide
    whether to run the build -->
  <modificationset[3] quietperiod="10"[4]>
   <cvs localworkingcopy="projects/MY_PROJECT_1"/>
  </modificationset>

  <!-- Configures the actual build loop, how often and which
    build file/target -->
  <schedule interval="60">[5]
   <ant[6] buildfile="build-MY_PROJECT_1.xml"
     uselogger="true"/>
  </schedule>

  <!-- directory to write build logs to -->
  <log/>[7]
```

```
  <!-- Publishers are run *after* a build completes -->
  <publishers>[8]
  </publishers>
 </project>
</cruisecontrol>
```

Explanation of configuration:

1: buildafterfailed="true"

The "buildafterfailed" property tells CruiseControl if it should keep on trying to build, even if the last time failed and there have been no changes. The good thing about this is that if the build failed because a database server was unavailable, or some other transient problem, the next attempt might succeed. The bad thing is that if there is a real problem with the source in version control, CruiseControl will just keep on trying to build until you commit a fix for the problem.

2: <currentbuildstatuslistener>

This element tells CruiseControl the name of a file where it can write notes about its current activity. This file is required by the reporting application, so we configure it here. For multi-project configs it is probably easiest to place it in the project logs directory, as above. This listener writes messages such as "Current Build Started At: [date]" to this file.

3: <modificationset>

This element configures how CruiseControl checks for changes in the version control repository, to determine if a build should be attempted. The example given tells CruiseControl to use CVS to check for changes in it's local working copy of the module which you checked out earlier.

4: quietperiod="10"

The "quietperiod" attribute tells CruiseControl not to try to build while the version control repository is being actively updated. Instead, CruiseControl waits until it sees a period of quiet in the repository before doing a build. If you're committing files individually to version control, CruiseControl will wait until you've finished, as long as you don't take longer than the specified quietperiod between commits. It is specified in seconds.

5: <schedule>

The <schedule> element sets up the build-loop for your project, with the "interval" attribute telling CruiseControl how many seconds to sleep in between polling for changes. With the interval="60" and quietperiod="10" attributes, CruiseControl will check for modifications in version control every 60 seconds. If modifications were made in the 10 seconds before the check, CruiseControl will wait until a 10 second window with no changes is found. Once this happens, CruiseControl will kick off the Ant build. These values are

probably a bit low for the real world, but they are OK for getting started.

6: <ant>
This element tells CruiseControl which Ant build file to run, and which target. The uselogger attribute tells CruiseControl to restrict what messages get written to the build log based on the Ant logging level. Since usedebug is false by default Ant debug statements won't be written to the build logs, making them much smaller. If you want to use your own installation of Ant you can do that by specifying the anthome attribute.

7: <log>
is element tells CruiseControl where to put its build logs, which are the main output of a build. The default location is "logs/[projectname]".

8: <publishers>
This element configures actions to perform after the build completes, such as sending emails, and copying files.

### *Start the Build Loop*

If you now run CruiseControl from WORK_DIR, you should see it start. CruiseControl will poll the version control system looking for changes and launching a build when if it finds some. The first time you run a project it will look for changes since midnight of the same day. If there haven't been any changes since then you'll need to commit a change to kick off a build. If the build fails, CruiseControl will keep on trying, until you get it right. You don't need to restart CruiseControl if you change config.xml or your delegating build file, since CruiseControl reloads these every time a build is performed.

If you look in WORK_DIR/logs/MY_PROJECT_1, you'll see the CruiseControl build logs, which are XML files. Any file that looks similar to log20081122345678Lbuild.1.xml indicates a successful build, while other XML files without the build label indicate failures. Fortunately, you don't need to parse these files yourself, because both the web reporting applications and the HTML Email publisher distributed with CruiseControl can present these results in HTML format. But for now, you're up and running, and every time you commit to version control, CruiseControl will pick up those changes and build them.

### *Reporting Build Status via Email*

A good way to keep track of your continuous integration is by receiving emails, either for every build, or just for the ones that fail. This is done by adding an <email> publisher to the set of publishers.

The most basic email functionality sends emails to one set of addresses on every single build (success or failure), and another set of addresses just on failed builds. To set this up, add an element like this inside the <publishers> element:

```
<email mailhost="SMTP_HOST"
      returnaddress="cc@mydomain.com"

buildresultsurl="http://localhost:8080/cruisecontrol/buildresults/MY_PROJECT_
1"
      skipusers="true" spamwhilebroken="true">
   <always address="dev1@mydomain.com"/>
   <always address="dev2@mydomain.com"/>
   <failure address="failed-builds@mydomain.com"/>
</email>
```

In this example, there are two addresses that will always receive build notifications, and another that will only receive notifications when the build fails.

If you also want individual committers to receive email for all builds where they made changes, then set skipusers="false", and add a <map alias="cvsuser" address="cvsuser@mydomain.com"/> element for each user inside the <email> element (replacing cvsuser with the real user id and email).

To get nicely formatted HTML mail you need to use the HTML email publisher instead of the plain email publisher. Replace <email> with <htmlemail> and add three extra attributes for css, xsldir and logdir. The complete <htmlemail> configuration looks like this:

```
<htmlemail mailhost="SMTP_HOST"
      returnaddress="cruise@mydomain.com"
      buildresultsurl="http://localhost/cc/buildresults/MY_PROJECT_1"
      skipusers="true" spamwhilebroken="true"
      css="INSTALL_DIR/reporting/jsp/webcontent/css/cruisecontrol.css"
      xsldir="INSTALL_DIR/reporting/jsp/webcontent/xsl"
      logdir="logs/MY_PROJECT_1">
   <always address="dev1@mydomain.com"/>
   <always address="dev2@mydomain.com"/>
   <failure address="failed-builds@mydomain.com"/>
</htmlemail>
```

## Running the Reporting Application

### *Installing the Reporting Application*

The CruiseControl Reporting Application is distributed as a J2EE web application archive (war) file, which can be found at INSTALL_DIR/reporting/jsp/dist/cruisecontrol.war.

It should be possible to install cruisecontrol.war on any J2EE Servlet Container. However, each container will has its own steps for installing the war. For example, using Tomcat, one way to install the war is to stop Tomcat, copy cruisecontrol.war to the directory TOMCAT_HOME/webapps, and then restart Tomcat. This will automatically extract and deploy the contents of the war.

Once the Reporting Application has been deployed, the Web Application Deployment Descriptor (web.xml) must be edited to allow the Reporting Application to find the CruiseControl log files and build artifacts.

Open the deployment descriptor from within the deployed location (for example, TOMCAT_HOME/webapps/cruisecontrol/WEB-INF/web.xml). The following two parameters will have to be modified as follows (make sure you replace WORK_DIR with the real working directory):

```
<param-name>logDir</param-name>
<param-value>WORK_DIR/logs</param-value>
<param-name>rootDir</param-name>
<param-value>WORK_DIR/artifacts</param-value>
```

Restart the Servlet Container and test the Reporting Application by opening a web browser and navigating to:

```
http://localhost:8080/cruisecontrol/index.jsp
```

### *Getting the Build Artifacts Link Working*

By using the artifacts publisher, together with the artifacts link in the Reporting Application, CruiseControl can provide access to historical build output, test results and other important build artifacts.

Add an <artifactspublisher> element to the <publishers> element of your config.xml, which publishes the desired build artifact(s) to a timestamped directory under the WORK_DIR/artifacts directory.

```
<artifactspublisher
   dir="projects/MY_PROJECT_1/build/output"
   dest="artifacts/MY_PROJECT_1"/>
```

This assumes that you want to publish all files that end up in the build/output directory after running the ant build. You can also use the file="..." form, but this unfortunately creates the entire directory structure under the artifacts dir, just to get the single file. Probably a better approach would be to modify your CruiseControl build to first copy the build artifacts to a common temporary location, so that your config file doesn't have to contain the path to the actual files in the checked-out project.

## Modifying the HTML Reports

The output you see in the Reporting Application, and the HTML emails, is the result of applying a number of XSLT stylesheets to the single XML build report that CruiseControl creates. You can see these reports in your WORK_DIR/logs directory.

By default, the log report is formatted like this:

```
<cruisecontrol>
  <modifications>
    (contains details of CVS changes since last build)
  </modifications>
  <info>
    (contains project details)
  </info>
  <build>
    (the XML output from ant)
  </build>
</cruisecontrol>
```

The header message you see on the web page is generated by an XSL stylesheet that reads the <info> element, and the "Modifications since last build" section is built by a stylesheet that uses the <modifications> element.

Various XSLT stylesheets are provided with the CruiseControl distribution, located in INSTALL_DIR/reporting/jsp/webcontent/xsl These include:

| Stylesheet | Purpose |
|---|---|
| header.xsl | Generates the build failed/success messages, and outputs the time of build and last changes. Uses the <info> and <modifications> elements. |
| modifications.xsl | Generates the "Modifications since last build" report from the <modifications> element. |
| compile.xsl | Looks for <javac> and <ejbjar> elements in your build |

| | |
|---|---|
| | output, and creates a report of the errors and warnings. |
| distributables.xsl | Builds a list of jars and wars generated by your build by searching the build output for <jar> and <war> tasks. |
| javadoc.xsl | Reports on errors and warnings generated by <javadoc> elements in your build. |
| logfile.xml | prints the entire XML log to HTML. This can be viewed in the XML Log File tab of the Reporting Application. |
| unittests.xsl | Builds a report on unit test failures, based on the presence of <testsuite> elements in your log file. These <testsuite> elements can be generated automatically by a <junitreport> task, but you must tell CruiseControl to merge these into the generated log file. Refer to the next section for an example of this. |
| checkstyle.xml | Builds a report of checkstyle failures, based on the presence of a <checkstyle> element in your log file. You must tell CruiseControl to merge your checkstyle output into your log file for this to work. Refer to the next section for an example of this. |

## Merging Other XML Files into the Logfile

In order for the "unittests" and "checkstyle" reports to work, you need to tell CruiseControl to merge the separate XML log files generated by <junit> and <checkstyle> into your main CruiseControl log file. This is done by adding a <merge> element to your <log> element in the config file.

```
<!-- directory to write build logs to -->
<log logdir="logs/MY_PROJECT_1">
   <merge dir="projects/MY_PROJECT_1/build/junit-reports/"/>
</log>
```

Note that you can have CruiseControl include a report from <junitreport> by using the file attribute (that is, <merge file="..."/>).

Once again, it may be better to copy any required files to a common temporary location in your build file, rather than coding the path to your checked out project in the config file.