

Part 2

Summary

This report continues the description of the automatic build system we use for building the Eclipse ECF plugin. It address the topics listed in the Fourth Stage and suggests topics for a Fifth Stage.

Contents

Summary.....	1
Additional Cruisecontrol Projects.....	2
Release Builds.....	3
Integration Build.....	3
The mapVersionTag	3
The Six Platform Plugins/Fragments.....	4
Plugins.....	4
Fragments.....	4
Jar Signing.....	4
createZipForSigning.....	4
signJars.....	4
signJarsReady.....	5
signJarsZip.....	5
deploy.....	5
Fifth Stage.....	5

Additional Cruisecontrol Projects

We now have nine cruisecontrol projects. The following table lists them. The extra projects result from the following needs.

- The Ganymede release uses ECF 2.0. Work on ECF must, of course, continue, and it does, being called 2.1. We made a CVS branch for ECF Release 2.0 called **Release_2_0**.
- The Eclipse Ganymede platform uses six plugin/fragments from ECF 2.0. Eclipse Ganymede is called 3.4. So when we build ECF 2.0, we must build two versions, one for Eclipse 3.3 (which contains those six plugins/fragments) and another for Eclipse 3.4 (which is missing those six plugins/fragments because they are already there as part of the platform). When we build ECF 2.1, it is always for Eclipse 3.4.

CC Project ¹	Build Type ²	Destination ³	Signed?	Ant Publisher	When run?	mapVTag ⁴	Map VersionTag ⁵
3.3ecf2.0	R33	2.0Test	yes	ant33Rel20.xml	forced 1PM	Release_2_0	Release_2_0
3.4ecf2.0	R34	2.0Test/3.4	yes	ant34Rel20.xml	forced 2PM	Release_2_0	p2_work around_1
ecf2.0	A20	not saved	no	none	30 min	Release_2_0	Release_2_0
ecfIntegration	I	integration	no	antint.xml	8AM Mon	Release_2_0	Release_2_0
3.3Daily2.0	D3320	dailies	no	antscp.xml	4PM	Release_2_0	Release_2_0
3.4Daily2.0	P20	3.4dailies	no	ant34Daily20.xml	3PM	Release_2_0	p2_work around_1
ecf2.1	A21	not saved	no	none	47 min	HEAD	HEAD
3.4Daily2.1	P21	3.4dailies2.1	no	ant34Daily21.xml	5PM	HEAD	p2_work around_1
3.3Daily2.1	D	dailies2.1	no	ant33Daily21.xml	6PM	HEAD	HEAD

1 Projects with 3.4 in the name are missing the six platform plugins/fragments.

2 The buildType is just an identifier used in build config files.

3 One dev.eclipse.org, prepend this directory with downloads/technology/ecf/. Note that the release builds go into test directories and must be manually copied into the release directories (2.0 and 2.0/3.4).

4 The mapVtag identifies the CVS branch. Calling it mapVTag seems a misnomer that has caused confusion.

5 The mapVersionTag identifies a CVS tag. Unless specified explicitly as something different, it equals mapVTag. Calling it mapVersionTag seems a misnomer. It operates like a CVS tag,

ECF Autobuild System

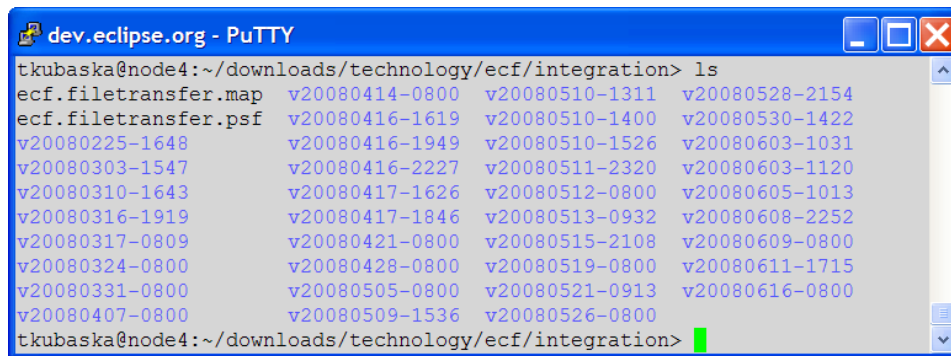
- The 2.0 builds monitor `/home/ted/workanonRelease_2_0` on `ecf2`. The 2.1 builds monitor `/home/ted/workanon` on `ecf2`.

Release Builds

Note that the two release builds are listed as forced. You can't have a strictly “forced” build in cruisecontrol. But what we do in these cases is set the modification set to a particular file rather than the CVS repository. Cruisecontrol checks the state of this particular file (`3.3daily_kick` for R33 and `3.4daily_kick` for R34) at 1PM for R33 and 2PM for R34. This means that if you touch one of those files, cruisecontrol will start build when its schedule has it look at them. But we never do modify those files, so effectively, the R33 and R34 are only forced builds.

Integration Build

The integration build provides plugins/fragments to the platform. Now actually we build everything and stick it in the `downloads/technology/ecf` directory. We tag this build as `v<timestamp>`. So what makes the integration build special is that we provide map and psf files that pull out the six plugin/fragments. We don't need to include all the other plugin/fragment jars, but it's just easier to do so.



```
dev.eclipse.org - PuTTY
tkubaska@node4:~/downloads/technology/ecf/integration> ls
ecf.filetransfer.map v20080414-0800 v20080510-1311 v20080528-2154
ecf.filetransfer.psf v20080416-1619 v20080510-1400 v20080530-1422
v20080225-1648 v20080416-1949 v20080510-1526 v20080603-1031
v20080303-1547 v20080416-2227 v20080511-2320 v20080603-1120
v20080310-1643 v20080417-1626 v20080512-0800 v20080605-1013
v20080316-1919 v20080417-1846 v20080513-0932 v20080608-2252
v20080317-0809 v20080421-0800 v20080515-2108 v20080609-0800
v20080324-0800 v20080428-0800 v20080519-0800 v20080611-1715
v20080331-0800 v20080505-0800 v20080521-0913 v20080616-0800
v20080407-0800 v20080509-1536 v20080526-0800
tkubaska@node4:~/downloads/technology/ecf/integration>
```

The mapVersionTag

Normally this is just HEAD or the name of the branch you're building. The exception is when you're building for 3.4. When we build for 3.4, we don't want to include the six platform plugins/fragments in the build product. But we need these plugins in our build space to make a successful build.

When we do a 3.4 build, we are building for the latest platform integration. So we want to take the copies of the six plugins/fragments used in that integration. We don't want to check out and build these plugin/fragments. So that's what the `ant_p2_workaround_1` tag does. It contains map files that leave out the six platform plugins/features.

We also made a branch called `ant_p2_workaround`. I don't believe we needed to do this; the tag was enough.

The key thing to note here is that a tag is not a branch. With a branch you can check out the branch and then make changes and check those changes into the branch without affecting the mainline. You have two separate lines of development.

ECF Autobuild System

A tag is not a separate line of development. It's a name for a collection of files. You can check out a tag, edit the files specified by the tag, check then in again and assign the same tag.

The Six Platform Plugins/Fragments

There are four plugins and two fragments. They are as follows.

Plugins

```
org.eclipse.ecf_2.0.0.v20080603-1120.jar
org.eclipse.ecf.filetransfer_2.0.0.v20080603-1120.jar
org.eclipse.ecf.identity_2.0.0.v20080603-1120.jar
org.eclipse.ecf.provider.filetransfer_2.0.0.v20080603-1120.jar
```

Fragments

```
org.eclipse.ecf.provider.filetransfer.ssl_1.0.0.v20080603-1120.jar
org.eclipse.ecf.ssl_1.0.0.v20080603-1120.jar
```

Jar Signing

We need to sign the two release builds: R33 and R34.

The signing is taken care of in the ant publishers: `ant33Re120.xml` and `ant34Re120.xml`. These files are very similar and may be combined at a later date. The ant task dependency is as follows.

```
deploy →
signJarsZip →
signJarsReady →
signJars →
createZipForSigning
```

createZipForSigning

When CC calls the ant publisher, it specifies the `deploy` task when then drops down to its lowest dependency, `createZipForSigning`, which does what it sounds like. It zips up the updatesite that was just built into a zip called `org.eclipse.ecf.updateJars.zip`. This zip contains just the updatesite except for the addition of the file `pack.properties`. The purpose of this file is primarily to determine the files that are not to be signed. In our case this is the two jars called `ch.ethz.iks.slp_1.0.0.RC2_v20080505-0900.jar` and `org.objectweb.asm_3.0.0.v200803061811.jar`.

```
ted@ecf2:~/Signed> cat pack.properties
pack200.default.args=-E4
pack.excludes=updateSite/plugins/ch.ethz.iks.slp_1.0.0.RC2_v20080505-0900.jar,updateSite/plugins/
org.objectweb.asm_3.0.0.v200803061811.jar
sign.excludes=updateSite/plugins/ch.ethz.iks.slp_1.0.0.RC2_v20080505-0900.jar,updateSite/plugins/
org.objectweb.asm_3.0.0.v200803061811.jar
ted@ecf2:~/Signed>
```

signJars

This task copies `org.eclipse.ecf.updateJars.zip` to a staging directory on build.eclipse.org.

ECF Autobuild System

Signing must be done on the machine `build.eclipse.org`. You have to have permission to run the signing script. The task waits until the signing is complete. After calling the signing script, it calls the target `waitForChangedAttribs`, which executes only if `attribs.changed` is not set.

The goal is to indicate signing completion by the setting of `attribs.changed`. So when the signing is complete, `waitForChangedAttribs` does not get called and the next operation is that the zip with now the signed jars comes back to `ecf2` in `/opt/build.ecf/ecf.signedOutput`.

```
waitForChangedAttribs calls compareAttribs, which sets the property
attribChanged if the signing has completed. Then compareAttribs calls
writeDiffResult if attribChanged is set. writeDiffResult creates the file
attribDiff.txt on ecf2. Then attribs.changed is set if attribDiff.txt exists,
waitForChangedAttribs does not get called, and the zip with the signed jars is sent
back to ecf2.
```

signJarsReady

This task just calls the bash script `getSignJars.sh`, which unzips the zip (with signed jars) that came back from `build.eclipse.org` into the updatesite directory. That `pack.properties` file came back as well, and the script deletes it.

signJarsZip

This task call the bash script `signZips.sh`. This script unzips the three zips that result from the build and lists the files in those zips. These files are not signed. The script deletes them and then copies the signed files from the update site using the list to ensure it gets the right files. Then it zips them up. The result is that the three zips resulting from the build now contain signed jars.

deploy

This task copies the signed zips and signed updatesite to `dev.eclipse.org`.

Fifth Stage

The major focus of our fifth stage will be to run some tests as part of the build.