European Laboratory for Particle Physics
Laboratoire Européen pour la Physique des Particules
CH-1211 Genève 23 - Suisse

Frisbee: FreeHEP Release Scheme

Document Date: 23 February 2001
Document Authors: Mark Dönszelmann (CERN)
Tony Johnson (SLAC)
Mario Ruggier (CERN)

Abstract

This document outlines *frisbee*, a proposed scheme for automatic deployment of software deliverables where the sources are managed in a CVS repository and the distribution environment is web-based. The deliverables, which may be anything from compiled libraries, executables, distribution kits, static web pages, as well as dynamic web applications, are built using the build tool Ant. Frisbee, as a layer above the build tool, automates the building of specified modules and releases as well as the deployment of specified publications and distributions.

1 Overview of the current situation

The FreeHEP Java Library[1] is an open source project to encourage the sharing and reuse of Java code in High Energy Physics. A number of packages, with a minimum of inter-dependencies, are offered to the HEP community of developers. The authors[1] of these FreeHEP packages are few, and available manpower is limited. A scheme to automate the precise procedural task of building new releases and deploying them for distribution is therefore needed.

The FreeHEP Java Library plus all other associated documents are managed in a CVS[2] repository. Once the project is checked out, the tool Ant[3] (a *make* in Java, and for Java) is used to compile the code, generate API documentation, prepare jar files, and collect everything in distribution files (in both zip and tar formats, for windows and unix) for the entire library.

The various public webs, primarily the ones for the FreeHEP library itself, HepRep[4], and for YaPPI[5], are also published directly, and automatically, from the CVS repository. This is done

1. http://java.freehep.org/
2. http://www.cvshome.org/
3. http://jakarta.apache.org/ant/
4. http://heprep.freehep.org/
5. http://yappi.freehep.org/

by means of a cron job that watches[6] the CVS repository for new check-ins which, when found, trigger a CVS update on the local copy of the files (this being the copy on the public web server) as well as invoking Ant to update API documentation and rebuild the jar files.

2 Features currently missing

1. A decoupling between a *cvs repository check-in* and *publishing* of an update to a web site or a sub-site.

   An author may wish to check-in material that is not ready to be published, for example to share it with other authors. The decision to publish a checked-in version of the web should require a human intervention, which should (a) be minimal and (b) not necessarily require the involvement of any other person other than the person doing the update. Furthermore, it should be possible to reverse back to any previous public web, irrespective of how many intermediate non-public check-ins have been affected in the meantime.

   Also, to allow proper testing prior to a decision to publish, a fully funtional development web (including all dynamic components) should be automatically built and available for review.

2. A similar automatic deployment of dynamic components of a web site, e.g. for a servlet requiring various source files to be compiled and brought together, and installed in a specific web server environment.

   For example, currently cvs updates to the YaPPI servlet must be propagated to the publicly available installation manually.

3. Individual distribution kits for independent modules.

   Currently only a distribution kit for the entire library is generated. Individuals wanting only sub-modules, e.g. YaPPI, must download the whole thing.

4. Incorporation of all deliverables into what constitutes a version of a web site.

   For example, the YaPPI deliverables could be a zip and a tar distribution kit. Publishing the YaPPI web should also include both the preparation of the distribution kits as well as their integration into the download area of the same web.

5. Possibility to recreate on demand earlier releases of a module, complete with the corresponding public web for the module.

3 The proposed release scheme

The general idea is based on using cvs tags to tag a release, in combination with a distribution configuration file, *distribute.frisbee*, for the cvs repoitory which is automatically processed by a

6. In fact, it just checks the CVS repository periodically for updates.

governing script, *frisbee*, to build the specified releases and deploy the specified distributions. A deliverable could be anything that may be defined as the output of an Ant target, such as compiled libraries, jar files, executables, distribution kits, static web pages, as well as dynamic web applications such as servlets or cgi scripts. Thus, even if we have concentrated this discussion mostly on web releases, this scheme should also be applicable to any other controlled deliverable.

3.1 Some additional requirements

1. End-users of the repository (those that check-out sources to build targets by running Ant with default parameter values) should not be affected by *frisbee*, i.e. the Ant layer (*build.xml* + property files) should be fully functional, independently of *frisbee*.
2. Different instantiations, or *distributions*, of essentially same releases (e.g. dev and pub releases) should be built using the exact same Ant targets and property values. The only possible differences (versions of source files and / or destination of the build) are stated in the *distribute.frisbee* file for the repository.
3. Ideally, a single generic *build.xml* file is maintained. Properties for modules to be built should be defined in property file for each module (these are much easier to define and maintain than *build.xml* files). Nonetheless, *frisbee* should also allow:
    a. the specification of a *custombuild.xml* file, if so desired.
    b. the addition of custom targets in the main *build.xml*, if so desired.

3.2 The overall process

The basic idea is that all updates to the CVS repository trigger an automatic dev release, while all *tagged* updates trigger automatically both a dev and a public release. Figure 1 illustrates this process, and individual steps are identified and explained below.

1. An author makes modifications to a checked out version of a web site or software module.
2. He may commit an update at any time.
   On the next scheduled checking for new updates, a *frisbee* governing script will do an automatic update of a dev release, by doing the following:
    a. checks out from CVS the latest versions of the main module, and sub-modules if any. Information such as the location of the CVS repository, the name of the main module[7], and so on are specified as parameters to the *frisbee* governing script (see Section 4, "The frisbee governing script").
    b. reads the *distribute.frisbee* file for the CVS repository (which would thus also be the latest, or *HEAD*, version of this file).

7. The file *distribute.frisbee* is part of the main module is contained in the main module. Names of any sub-modules to check out may be determined from the *distribute.frisbee* file.

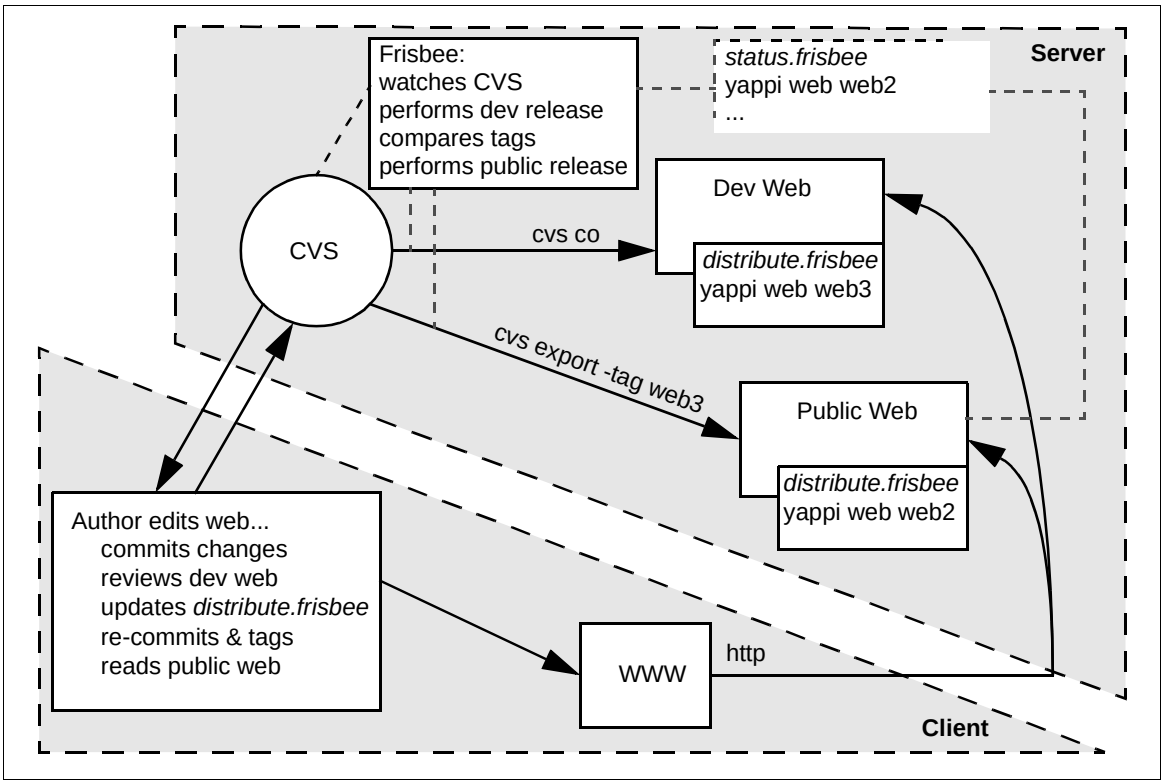

Figure 1 Overview of the freehep release scheme, *frisbee*.

c. invokes Ant to rebuild and deploy all dev targets (those that have <cvstag> set to "*") that are affected by the newly commited changes in the sources, using destination information[8] that is also specified in *distribute.frisbee*.

The script will also check to see if information in the *distribute.frisbee* file has changed with respect to what is currently published at that point in time. This is done by keeping a status file of whatever is published at any given time (see Section 4.1, "Frisbee script logic").

As long as the information in the *distribute.frisbee* is equivalent to the information in the status file, the script will stop here, doing nothing on any public (tagged) versions.

3. To affect a public release, i.e. to publish a tagged release, an author would:
    1. edit the *distribute.frisbee* file for the CVS repository, specifying all details of what is to be published, and where (see Section 3.4, "Syntax of the distribute.frisbee configuration file").
    2. do a cvs commit with all updated files, including the modified *distribute.frisbee* file.
    3. assign a cvs tag to this release, using the same cvs tag as that just specified in the *distribute.frisbee* file.

8. Property files for each module should contain default values for all build destinations, so that doing "ant" on a generic "cvs co" (outside of *frisbee*) will build correctly the latest versions for all targets.

These 3 steps can be summarized as follows:

```
edit distribute.frisbee
cvs commit
cvs tag <tag>
```

As in step (2), this new cvs commit[9] will also trigger an automatic update of a dev version of the web.

The *frisbee* script will again check to see if the *distribute.frisbee* file has changed with respect to the status file of what is published at this point in time.

If changes are found such that a public web or distributions should be updated, then the frisbee script should do exactly that, namely to check out the tagged releases specified in the *distribute.frisbee* file, and then build each target by running Ant.

3.3 A little example

Let's take a look at a minimal example of how a build and distribution of a module is specified in the *distribute.frisbee* file. The configuration lines required are shown in Listing 1.

Listing 1 The configuration lines required in *distribute.frisbee* to specify which tagged version of the freehep web to publish, and where.

```
freeheppub    *.module    = freehep
freeheppub    web.dst     = /afs/slac/freehep/web
freeheppub    web         v2r0p0doc
```

The first line sets a variable, **.module* to "freehep"[10], that applies to all targets of the *freeheppub* distribution. The second line sets another variable, *web.dst*, that applies to the target *web* of the *freeheppub* distribution. The third line specifies an action to be performed, precisely to build version *v2r0p0doc* of the *web* for *freeheppub* distribution (using module "freehep", and putting the results in *web.dst*, i.e. "/afs/slac/freehep/web").

After *frisbee* reads these lines (after each default *HEAD* check out that it performs) it will:

1. do a tagged cvs export, using "v2r0p0doc" as the cvs tag, in an export area (specified as a script parameter), using module "freehep":

   ```
   cd <work area>
   cvs export -r v2r0p0doc freehep
   ```

9. CVS should be set-up such that a flag is set each time a "cvs commit" or a "cvs tag" is performed, e.g. by means of touching a file called *dofrisbeeupdate*. There could still be discrepancies between tags referred to in *distribute.frisbee* and those defined in the CVS repository, e.g. incorrectly spelt tags, or a tag is defined in CVS after a commit (that refers to it) has been done. The script will therefore receive a cvs error when trying to do the tagged check-out, as the tag does not exist, or at least not yet. In this case the script should log the error and either try again after a fixed time period, and/or send a mail to a responsible (maybe the individual author who affected the cvs commit?) to inform of the problem.

10. We need to know the module name to (a) know what module to check out, if not yet done and (b) to know what property file to use when building targets for this distribution.

2. run Ant with target “web”, specifying *web.module* and *web.dst* as properties on the commandline:

   ```
   ant -Dweb.module=freehep -Dweb.dst=/afs/slac/freehep/web web
   ```

The rest of the information needed to build and deploy the *freehep web* is obtained from the file *build.xml* (and its definition of target *web*) and from a *freehep.properties* file, if any.

3.4 Syntax of the *distribute.frisbee* configuration file

This simple file specifies, for each module, which deliverables should be built and published to the download and web area, e.g. which web release, which apidoc, and which distribution kits should be made available. Only two kinds of declarations are allowed, these being (a) setting variables and (b) building targets. The syntax is shown in Listing 2, and explanation of each keywords follows.

Listing 2 The general format of a *distribute.frisbee file*. Any number of releases, targets, or variables may be defined. Development and public releases are simply treated as different releases.

```
# comment
<distribution>   <target>.<param> = <value>            # variable definitions
<distribution>   <target> <cvstag> [<version>]         # target specification
```

<distribution> An arbitrary string to associate variables and targets of the same distribution. For variable declarations, a value of “*” may be assigned, which would associate this *variable* (in the form of *target.param*) to the target *target* of all *distribution*s.

<target> Is a valid target defined in Ant’s build.xml file. For variable declarations, a value of “*” may be assigned, which would associate this *variable* (or *target.param*) to all *target*s (for this *distribution*).

<param> An arbitrary parameter name and part of the variable name. When building *target* for *distribution*, variable values are passed to Ant as commandline properties in the form of -D*target.param*=<value>.

<cvstag> A tag used on the CVS repository or module, to tag a release. If a value of “*” is spcified, then the latest version of sources in CVS will be used.

<version> An arbitrary string (but a valid file name). If specified, then a subdirectory *version* is created, at the destination location for the build, e.g. below the location specified in an associated *target.dst* variable.

3.5 Other conventions assumed by *distribute.frisbee*

1. Different build destinations (e.g. latest dev release and a public tagged release) are simply treated as different physical distributions.
2. In variable definitions (and only in variable definitions!) both *distribution* and *target* parts could be set to "*". This will be interpreted as "all distributions" or "all targets", respectively, unless otherwise redefined for specific distributions or targets. E.g.

   ```
   fhpub      *.module    = freehep
   fhpub      web         v2r0p0doc
   fhpub      apidoc      v2r0p0
   fhpub      dist        v2r0p0
   ```

   implies a variable definition for each target associated to the *fhpub* distribution, i.e.:

   ```
   (fhpub) web.module
   (fhpub) apidoc.module
   (fhpub) dist.module
   ```

   Furthermore, something like:

   ```
   *          *.module    = freehep
   fhpub      web         v2r0p0doc
   yappi      web         v2r0p0
   ```

   would imply a variable per target and per distribution, i.e.:

   ```
   (fhpub) web.module
   (yappi) web.module
   ```
3. For each target, *frisbee* assumes a *target.clean* that states what is needed to be done to remove *target*.
4. *Frisbee* will look for the file *distribute.frisbee* in the top level directory of the frisbee cvs module (this can ofcourse be the main module of teh repository).

3.6 A sample *distribute.frisbee* file

A sample *distribute.frisbee* file is shown in Listing 3. All cvs updates will trigger the script to build all dev targets (those with a "*" as <cvstag>), by running ant with the the associated variables values. Tagged cvs updates (that also include changes to the *distribute.frisbee* file) will also trigger the script to build all non-dev targets, again running ant but using the different set of variable values.

On a default check-out, *frisbee* will build the *fhdev* and *ydev* distributions. Note that it will put the distribution for *ydev* in a subdirectory that it will create, called "latest", below "/afs/slac/yappi/dev/download".

If *distribute.frisbee* has changed (when compared to *status.frisbee*), then frisbee will also build the tagged distributions *fhpub* and *ypub*. Note that *ypub* redefines the *module* variable from "freehep" to "yappi". Another interesting point is that 3 distributions will be built for *fhpub*, and put in 3 different subdirectories. Similary, 2 versions of the *ypub* web will be built. Furthermore, *ypub* does not rebuild *apidoc*, which has already been built in *fhpub*.

Listing 3 A sample *distribute.frisbee* file for releases originating from the FreeHEP CVS repository.

```
# Default module name
*         *.module     = freehep
*         *.antrundir  = freehep

#### freehep public ####
fhpub     web.dst      = /afs/slac/freehep/web
fhpub     apidoc.dst   = /afs/slac/freehep/web/apidoc
fhpub     dist.dst     = /afs/slac/freehep/web/download
fhpub     web          v2r0p0doc
fhpub     apidoc       v2r0p0         2.0
fhpub     dist         v2r0p0         2.0.0
fhpub     dist         v1r2p6         1.2.6
fhpub     dist         millenium      m

#### freehep dev  ####
fhdev     web.dst      = /afs/slac/freehep/dev
fhdev     apidoc.dst   = /afs/slac/freehep/dev/apidoc
fhdev     dist.dst     = /afs/slac/freehep/dev/download
fhdev     web          *
fhdev     apidoc       *
fhdev     dist         *

#### yappi public ####
ypub      *.submodule  = yappi
ypub      web.dst      = /afs/slac/yappi/web
ypub      war.dst      = /afs/slac/yappi/web/tomcat/webapps
ypub      dist.dst     = /afs/slac/yappi/web/download
ypub      web          v1r0p0doc
ypub      web          v0r2p0doc      0.2.0
ypub      war          v2r0p0
ypub      dist         v1r0p0         1.0.0

#### yappi dev  ####
ydev      *.submodule  = yappi
ydev      web.dst      = /afs/slac/yappi/dev
ydev      war.dst      = /afs/slac/yappi/dev/tomcat/webapps
ydev      dist.dst     = /afs/slac/yappi/dev/download
ydev      web          *
ydev      war          *
ydev      dist         *              latest
```

4 The frisbee governing script

Frisbee may be called from a cron job or from a regular shell script. In either case several command line parameters will be required:

```
frisbee -c  $CVSROOT
        -f  $FRISBEEMODULE
        -w  $WORKLOCATION
        -e  $EXPORTLOCATION
        -s  $STATUSLOCATION
        -l  $LOGLOCATION
        -u  $UPDATEFILEFLAG        # Optional
```

Some online help will be offered when doing:

```
frisbee -help
```

4.1 Frisbee script logic

1. If -updatefileflag switch is specified, frisbee first determines if an update is necessary by checking existence of file $UPDATEFILEFLAG.
2. Checks out the latest version of the sources:

   ```
   cd $WORKAREA
   cvs -d $CVSROOT co $FRISBEEMODULE
   ```

3. Reads *distribute.frisbee*, located at:

   ```
   $WORKLOCATION/$FRISBEEMODULE/distribute.frisbee
   ```

4. Performs the dev releases:

   Runs Ant for each target found in *distribute.frisbee* for which <cvstag> version is “*”, passing the associated variable values as properties set on the Ant command-line.

5. Compares the latest *distribute.frisbee* with $STATUSLOCATION / *status.frisbee*, and:

   Tagged targets with changed information
   For each non-”*”<distribution> <target> line found, compares the <cvstag> (i.e. the cvstag of the current public version of <distribution> <target>) with the one in the *status.frisbee* file. If the <cvstag> is different, then the script will do a:

   ```
   cd $EXPORTLOCATION
   cvs export -r <cvstag> <module>
   ```

   Performs Ant building in exactly the same way as it did for dev distributions.

   If building successful, it updates the *status.frisbee* file.

   Logs all changes to a separate accumulative $LOGLOCATION / *frisbee.log* file.

   Target that have disappeared
   Each tagged distribution item in the *status.frisbee* file that is not found in the latest *distribute.frisbee* file will be *unpublished*, or removed from distribution. This is handled by assuming, for each *target*, a definition of a “*target*.clean” target in the *build.xml* file, stating what is required to delete *target*.

4.2 Build.xml and property files

As stated earlier, the Ant layer should be fully functional independently of *frisbee*. Furthermore, we would hope to configure all targets using a generic *build.xml* plus as many custom property files as necessary. In this way, the *build.xml* file could be used across modules in the same respository, as well as across projects in different repositories.

The FreeHEP build requirements could be met by the set of generic targets shown in Listing 4.

Listing 4 Primary targets defined in build.xml, that should satisfy all of FreeHEP requirements.

```
all        <   default> builds: compile, rmi, jni and jar
dist       <gen-target> creates distribution kits
apidoc     <gen-target> creates apidoc
help        <    target> display help on how to define specific targets
war        <war-target> creates war file
web        <web-target> collects all components of web
```

What Ant will do for each target will depend on the the variable settings in *distribute.frisbee*, and on values of other properties in the property file for the module. For example, to build:

```
ypub    *.module      = yappi
ypub    web.dst       = /afs/slac/yappi/web
ypub   web          v1r0p0doc
```

Frisbee will run first run ant with:

```
ant -Dweb.module=yappi -Dweb.dst=/afs/slac/yappi/web web
```

Ant will look for a *build.xml* file, in which it will find references to property files to include, from which other properties for *module* are obtained:

```
<property file="config/${module}.properties" />
```

The *web* target could specify things such as:

```
copy ${webdir}/${module}/* ${web.dst}/*
```

While the *web.clean* target could specify:

```
remove ${web.dst}/*
```

A The YaPPI public web

The yappi public web file system could like like this:

```
http://yappi.freehep.org/

    index.html                contains also links to external resources:
                              jCVS for source code browsing
                              yappi servlet (may be somewhere else physically)
                              to yappi apidocs within full freehep apidocs
                              download area

    images/                   may contain other resources than just images,
                              e.g. css, js, ...
                              should have a more generic name, e.g. assets/
    reports/ ...
    talks/ ...
    dev/
        UserRequirements.html
        classdiagram.ppt
        ...
    user/...

    download/[11]              points to a directory (a la jakarta distribution)
        README                all in here is cvsignored except for README
        yappi-version.tar(s)
        yappi-version.zip(s)
       yappi.war?, yappi.jar? yappi-javadoc.zip?

    servlet/             may physically be somewhere else but URL to use to link
                         to it must be known beforehand, and should be relatively
                         identical in both dev and public webs.
```

What a user will perceive in his web browser is different from this. The organization of this information is presented primarily in two categories:

a. YaPPI user

b. YaPPI developer, FreeHEP maintainer

It would be nice to optimize the different web areas to connect more easily together:

- Configure page templates (wherever possible) for apidocs, jCVS, servlet, to include sensible links back to base, include copyright info, and so on.
- For generated independent blocks (for which page layout is not really configurable), maybe a secondary window should be launched, so that the user will not lose the initial context. This could be for apidocs, jCVS, and download areas.

11. Should the download area for each module be in the same diretory tree as its web. or should there be a central freehep download area, organized by module? The latter would enable easy browsing from the downlaod area of a given module to that of any other.