

## **Table of Contents**

Introduction.....	2
Other Resources.....	2
Guiding Principles and Best Practices.....	2
Overview of the NCICB Build Environment.....	2
Build scripts should not interact with CVS.....	3
Keep build scripts simple.....	3
Projects should be self-contained.....	3
Build scripts must run unattended.....	3
Builders do not edit property files.....	3
Managing dependencies.....	4
Make use of Ant's filtering capability.....	4
Externalize property files.....	4
Providing Feedback.....	4

## **Introduction**

The purpose of this document is to provide some guidelines for writing build scripts to support the goals of the NCICB Software Configuration Management Initiative and the NCICB Open Source Initiative.

Although most development teams are currently using Ant as a build tool, the build server (Anthill Pro) can also support Maven-based scripts. Teams that are interested in using Maven are encouraged to contact the NCICB SCM Administrator to discuss specifics.

This document is not intended to be a complete tutorial on Ant or Maven; there are a number of online resources that do that already (a partial list is included below). Teams are strongly encouraged to implement the guidelines described here and/or suggest improvements or additional content.

As always, the SCM Administrator is happy to assist with build scripts in any way possible.

## **Other Resources**

The following is a partial list of resources that may be useful. In addition, a number of books are available through either Amazon.com or Barnes & Noble.

- <http://ant.apache.org/>
- <http://wiki.apache.org/ant/FrontPage>
- <http://maven.apache.org/>
- <http://www.urbancode.com/products/anthillpro/default.jsp>

## **Guiding Principles and Best Practices**

### ***Overview of the NCICB Build Environment***

The SCM Initiative build server (AnthillPro) is on a physically separate machine from the deployment servers. Therefore tasks or goals that copy files to a deployment directory *cannot* be used in the build environment, and your script should not expect that any of your project's software dependencies are installed on the build machine.

Building and deployment are two logically separate tasks. The output of the build process is a deployment unit (a deployable archive or executable jar, plus all necessary supporting files) that will be handed off as a unit to the Deployment team for deployment to the target application server. Thus every build script must support a single target or goal that provides the necessary deployment unit.

***Build scripts should not interact with CVS***

It is not necessary to define tasks that update from CVS before a build. AnthillPro manages all interactions with CVS for the purposes of staging and production builds. *No exceptions to this rule will be made.* The proper tag will be checked out before the build is started.

***Keep build scripts simple***

The more complex the build process, the harder it will be to maintain the script. Build larger targets from smaller, special purpose targets, using the dependency mechanism in Ant or the preGoal and postGoal mechanisms in Maven. Complicated build scripts are almost always a sign of trouble.

***Projects should be self-contained***

Build scripts should not (for the most part) rely on any particular configuration details of the build machine, beyond requiring a particular JDK and version of Ant. It should be possible to check out a project on any machine with the specified JDK and Ant version, and execute at the minimum a target that creates a deployment unit, for a specified tier (dev, qa, staging, production). This is important because the build servers will, in general, not be reconfigured to accommodate project idiosyncrasies.

A good test of whether your build script truly is self-contained is to check it out on an arbitrary machine (preferably, one that has nothing other than the operating system, and empty classpath, and the particular versions of the JDK and Ant that your script expects, not one that has already been heavily modified for development purposes), check out your project, and execute your build task. If the build works under those circumstances, it's likely that it will work properly on the build server as well.

Additionally, it's generally not a good idea to rely on specific environment properties to control builds. Ant and Maven provide a number of mechanisms to allow users to specify properties at runtime and these mechanisms should be explored before environment properties are considered.

***Build scripts must run unattended***

Any target that halts to wait for user input will eventually time out on Anthill. If your script expects user input, make provisions to allow it to be specified in some alternative way (such as via a property on the command line).

***Builders do not edit property files***

Build instructions that require builders to edit property files are not acceptable. If your build script requires properties to be set at build time, set up your script in such a way as to allow those properties to be specified via the command-line, rather than by editing a

file. **This does not mean that all of your project's mutable properties should be specified on the command line.** It is entirely possible (and encouraged) to write build scripts that allow arbitrary properties to be overwritten by values provided on the command line. This is most easily accomplished using filters along with the copy tasks in Ant. Consult the documentation, or discuss specific examples with the SCM Administrator ([scmadmin@mail.nih.gov](mailto:scmadmin@mail.nih.gov)).

### ***Managing dependencies***

AntHill has a fairly sophisticated build dependency management capability. If your project has a dependency on another project, AntHill will manage most of the heavy lifting automatically. Essentially, dependencies are managed through properties specified at build time. A dependency build script is passed a property that tells it where it should copy its output. The build script simply needs to take steps to copy all the files that the dependent project requires to the location specified in the property. The dependent project's build script will also be passed a property, telling it where to find all the artifacts that its dependencies have generated. Those artifacts can then be added to the classpath or otherwise manipulated through that property in the dependent project's build script.

### ***Make use of Ant's filtering capability***

An excellent and flexible approach to writing portable build scripts revolves around the use of Ant's filtering capabilities, using the filter task in combination with the copy or move tasks. The key is to define your project's mutable properties in a build.properties file that is used as a source of filters, whose values are inserted into the project's runtime property files. Such an approach allows builders to set default properties in the build.properties file, while also allowing them to be easily overridden on the commandline, so that, for example, database settings can be specified at build time.

### ***Externalize property files***

Property files that are external to the deployment unit are much easier to customize than embedded property files, and can speed up diagnosis and repair of non-working software. There are a number of strategies for making external property files available within an application. The Systems team or SCM Administrator can be of assistance here.

## **Providing Feedback**

Please send all feedback to the SCM Administrator at [scmadmin@mail.nih.gov](mailto:scmadmin@mail.nih.gov).