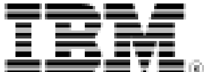


developerWorks.

Automation for the people: Manage dependencies with Ivy

Use a common repository with Apache Ant to share other projects' source code

Skill Level: Introductory

Paul Duvall (paul.duvall@stelligent.com)
CTO
Stelligent Incorporated

06 May 2008

Managing source-code dependencies among projects and tools is often a burden, but it doesn't need to be. In this installment of *Automation for the people*, automation expert Paul Duvall describes how you can use the Apache Ant project's Ivy dependency manager to handle the myriad dependencies that every nontrivial Java project must manage.

Virtually every software-development project must depend on the source code from other projects. For instance, many of your projects probably rely upon a logging utility such as log4j or a Web framework such as Struts. Your development team doesn't maintain those other projects' source code, yet you rely on their APIs to implement your project's custom software. The greater the number of other projects your software depends on, including any of those projects' own dependencies, the more complex building your software becomes.

About this series

As developers, we work to automate processes for users; yet, many of us overlook opportunities to automate our own development processes. To that end, *Automation for the people* is a series of articles dedicated to exploring the practical uses of automating software development processes and teaching you *when* and *how* to apply automation successfully.

I've found that teams use various imperfect techniques to try to solve this dilemma:

- Placing all dependent projects (JAR files) in a directory that's checked into

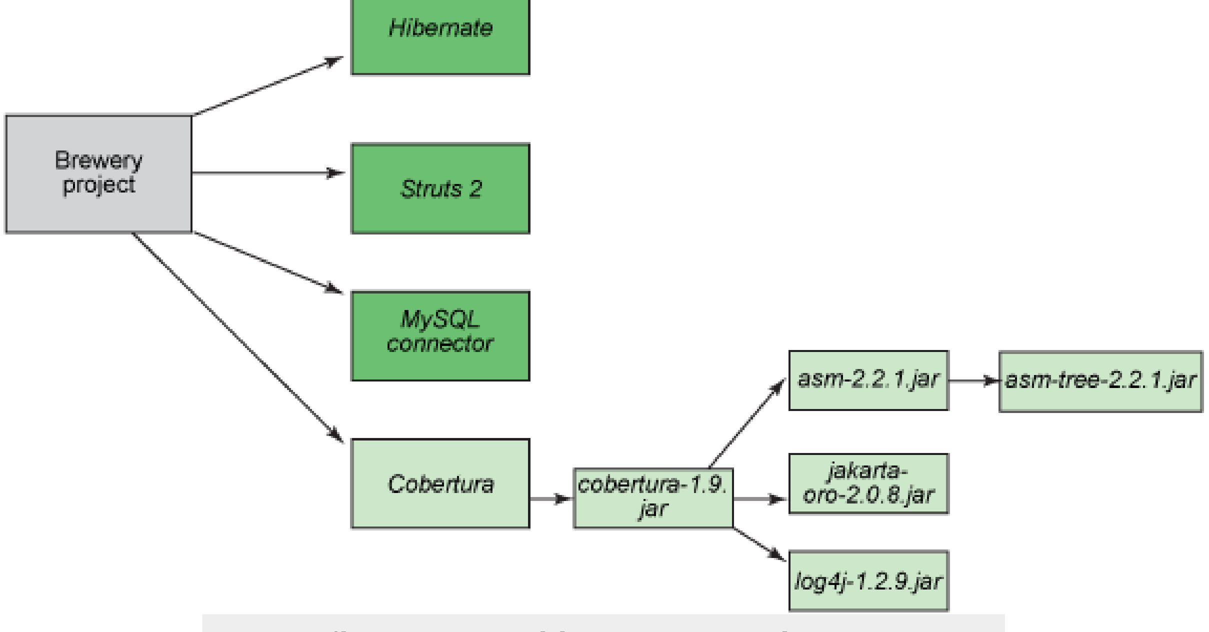
© Copyright IBM Corporation 1994, 2007. All rights reserved.

the project's version-control repository. This technique bloats the repository unnecessarily, making it difficult to manage version differences.

- Allocating dependent JARs to a common file server, which prevents the team from controlling version changes.
- Copying JAR files manually to a specific location on each developer's workstation. This approach makes it difficult to determine missing files or correct versions.
- Performing an HTTP `Get` to download files to a developer's workstation, either manually or as part of the automated build. This technique requires duplicate scriptlets and often leads to unmanaged JAR files.

I worked on a medium-sized project that contained 1,000 Java classes and more than 100 dependent JAR files. (We chose the first imperfect technique: checking in each of these JARs to the project's version-control repository.) Figure 1 shows a small fraction of the types of dependencies you can expect to see in a such a project:

Figure 1. Example dependencies for JARs in a Web development project



Transfixed on transitive dependencies

Transitive dependencies is a big phrase for a simple yet powerful feature provided by Ivy. Some JAR files depend on other JARs in order to work correctly. With Ivy, you declare a component's dependencies only once. From that point on, you need to know only the main JAR file for a project rather than all of its underlying JAR file dependencies. If you've experienced the pain in manually chasing down dependencies either through documentation or by investigating code, you'll find that this feature alone is worth the time in configuring Ivy in your project. See Depending on dependencies in this article for details.

Figure 1 illustrates that the Brewery project's source code depends on Hibernate,

© Copyright IBM Corporation 1994, 2007. All rights reserved.

Struts 2, MySQL Connector, and Cobertura. In turn, Cobertura depends on other JARs, such as asm-2.2.1.jar, jakarta-oro-2.0.8.jar, and log4j-1.2.9.jar. Further, asm-2.2.1.jar depends on asm-tree-2.2.1.jar. This is merely a simple example of the types of nested dependencies that can occur. If even one of the JAR versions is incorrect, you can experience problems that are difficult to troubleshoot, such as compilation errors or unexpected behavior.

The Apache Maven build- and project-management tool has gained some traction among Java developers. Maven introduces the concept of a common repository of JAR files accessible through a publicly available Web server (called ibiblio). The Maven approach reduces the JAR-file bloat that consumed most version-control repositories. But using Maven encourages you to adopt its "convention over configuration" approach to building software, which can limit your flexibility in customizing your build scripts.

What if you've been using Apache Ant for years and want the benefits of using a common repository? Are you forced into accepting Maven's build approach to get these benefits? Fortunately, the answer is no, because of a tool called Apache Ivy — an Ant subproject. Ivy offers a more consistent, repeatable, and easier-to-maintain approach to managing all of your project's build dependencies (see Resources for a comparison between Maven and Ivy). This article covers the basics of installing and configuring Ivy to manage dependencies and points you to additional information you can examine in greater detail.

Getting started

Getting started with Ivy is as simple as creating two Ivy-specific files and adding a few Ant targets. The Ivy-specific files are ivy.xml and an Ivy settings file. The ivy.xml file is where you list all of your project's dependencies. The ivysettings.xml file (you can name this file anything you wish) is where you configure repositories that the dependent JAR files will be downloaded from.

Listing 1 shows a simple Ant script that calls two Ivy tasks: `ivy:settings` and `ivy:retrieve`:

Listing 1. Simple Ant script using Ivy

```
<target name="init-ivy" depends="download-ivy">
  <ivy:settings file="${basedir}/ivysettings.xml" />
  <ivy:retrieve />
</target>
```

In Listing 1, `ivy:settings` defines the Ivy settings file. The call to `ivy:retrieve` retrieves the JAR files from one of the repositories declared in ivy.xml.

Installing Ivy

You have a couple of options for downloading and using Ivy. The first is to download the Ivy JAR file manually to your Ant lib directory or to a different directory that you

© Copyright IBM Corporation 1994, 2007. All rights reserved.

define in your Ant script's classpath. I'm a fan of automation, so I prefer the automatic alternative: downloading Ivy's JAR and configuring the classpath in Ant targets. Listing 2 shows an example of this technique:

Listing 2. Using Ant to install Ivy automatically

```
<?xml version="1.0" encoding="iso-8859-1"?>
<project name="test-ivy" default="init-ivy" basedir="."
   xmlns:ivy="antlib:org.apache.ivy.ant" xmlns="antlib:org.apache.tools.ant">
  <property name="ivy.install.version" value="2.0.0-beta2" />
  <property name="ivy.home" value="${user.home}/.ant" />
  <property name="ivy.jar.dir" value="${ivy.home}/lib" />
  <property name="ivy.jar.file" value="${ivy.jar.dir}/ivy.jar" />

  <taskdef resource="org/apache/ivy/ant/antlib.xml"
     uri="antlib:org.apache.ivy.ant" classpath="${ivy.jar.dir}/ivy.jar"/>

  <target name="download-ivy">
    <mkdir dir="${ivy.jar.dir}"/>
    <get src="http://www.integratebutton.com/repo/
       ${ivy.install.version}/ivy-2.0.0-beta2.jar"
      dest="${ivy.jar.file}" usetimestamp="true"/>
  </target>

</project>
```

The second line in Listing 2 defines the XML namespace. `antlib` refers to a file called antlib.xml in the ivy.jar file. The rest of the `xmlns` indicates the fully qualified path for the `ivy` Ant task. The `ivy.home` value of `${user.home}/.ant` is the location where the ivy.jar file will be downloaded. The `taskdef` defines the `ivy` Ant task, referring to its classpath location. The `download-ivy` target downloads ivy-2.0.0-beta2.jar and renames it using the `dest` attribute.

Once you download and configure Ivy, you can use any of the Ivy Ant tasks (such as the two tasks called in Listing 1).

Creating configuration scripts

The ivy.xml file, where you define all of your project's dependent JARs, is required. Listing 3 shows an example:

Listing 3. Defining dependencies in ivy.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="./config/ivy/ivy-doc.xsl"?>
<ivy-module version="1.0">
  <info organisation="com" module="integratebutton" />
  <dependencies>
    <dependency name="hsqldb" rev="1.8.0.7" />
    <dependency name="pmd" rev="2.0" />
    <dependency name="cobertura" rev="1.9"/>
    <dependency name="checkstyle" rev="4.1" />
    <dependency name="junitperf" rev="1.9.1" />
    <dependency name="junit" rev="3.8.1" />
  </dependencies>
</ivy-module>
```

Notice that Listing 3 includes no indication of file locations or URLs, letting you move

© Copyright IBM Corporation 1994, 2007. All rights reserved.

to different repository locations without needing to change the list of dependencies. The `organisation` attribute in the `info` element identifies the organization type (such as .net, .org, or .com). This is followed by the `module` name. The list of dependencies for this module follows a naming convention that will become more clear in the next listing. For now, just remember that `dependency name="cobertura" rev="1.9"` translates to cobertura-1.9.jar.

Listing 4 is an example of a Ivy settings file. It defines the repository locations and associated patterns used in the ivy.xml file in Listing 3.

Listing 4. Ivy settings file

```
<ivysettings>
  <settings defaultResolver="chained"/>
  <resolvers>
    <chain name="chained" returnFirst="true">
      <filesystem name="libraries">
        <artifact pattern="${ivy.conf.dir}/repository/[artifact]-[revision].[type]" />
      </filesystem>
      <url name="integratebutton">
        <artifact pattern="http://www.integratebutton.com/repo/[organisation]/[module]/
           [revision]/[artifact]-[revision].[ext]" />
      </url>
      <ibiblio name="ibiblio" />
      <url name="ibiblio-mirror">
        <artifact pattern="http://mirrors.ibiblio.org/pub/mirrors/maven2/[organisation]/
           [module]/[branch]/[revision]/[branch]-[revision].[ext]" />
      </url>
    </chain>
  </resolvers>
</ivysettings>
```

The `filesystem` element in Listing 4 defines the location pattern on your local workstation. The two `url` elements define the multiple locations the JAR files can be downloaded from: the first defines a custom repository at integratebutton.com that I control, and the next defines an external Maven repository (not under my control) that contains a plethora of open source JAR files. If Ivy is unable to download from the first repository — if it's down, for example, or if the file isn't in the specified location — it tries the next one. The beauty is that once Ivy downloads a JAR, it puts the file on your local file system so that it doesn't need to download these files for every build.

Depending on dependencies

It's typical for a module to have dependencies on other modules. In Figure 1, for instance, you saw that the cobertura-1.9.jar file's multiple dependencies include asm-2.2.1.jar, and that asm-2.2.1.jar has a dependency on asm-tree-2.2.1.jar. Without a tool like Ivy, you would need to ensure the correct versions of these JARs are in the classpath and that no conflicts exist among JAR versions. With Ivy, you simply define the `cobertura` module and all of its dependent modules, as shown in the example ivy.xml file in Listing 5. Keep in mind that this ivy.xml file resides in the same directory as the cobertura-1.9.jar file.

Listing 5. Defining dependencies in ivy.xml

© Copyright IBM Corporation 1994, 2007. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<ivy-module version="2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">
  <info organisation="cobertura" module="cobertura" revision="1.9"/>
  <configurations>
    <conf name="master"/>
  </configurations>

  <publications>
    <artifact name="cobertura" type="jar" conf="master" />
  </publications>

  <dependencies>
    <dependency org="objectweb" name="asm" rev="2.2.1" conf="master"/>
    <dependency org="jakarta" name="oro" rev="2.0.8" conf="master"/>
    <dependency org="apache" name="log4j" rev="1.2.9" conf="master"/>
  </dependencies>
</ivy-module>
```

The highlighted dependency in Listing 5 defines the `objectweb org` and the name `asm` along with the particular revision to use. Ivy uses this information along with the repository definitions in the ivysettings.xml file (as shown in Listing 4) to download the JAR file dependencies.

Figure 2 illustrates the directory structure from a repository that conforms to the configuration in the ivysettings.xml file from Listing 4:

Figure 2. Directory structure for asm module

Notice also that Figure 2 shows an ivy.xml file (described in Listing 6) that defines the dependencies for `asm`. In Listing 6, a snippet of the ivy.xml for the `asm` module demonstrates its single dependency — asm-tree-2.2.1.jar:

Listing 6. ivy.xml to define dependencies for asm

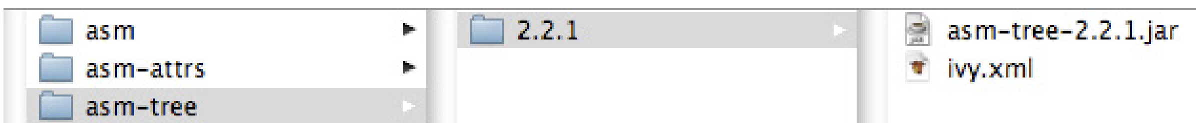
```
...
<dependencies>
  <dependency org="objectweb" name="asm-tree" rev="2.2.1" conf="master"/>
</dependencies>
...
```

To recap, the `cobertura` module defines three dependent modules: `asm`, `jakarta-oro`, and `log4j`, as shown in Listing 5. In turn, the `asm` module has a single dependent module called `asm-tree`, as shown in Listing 6.

Notice how similar the directory structure for `asm-tree` in Figure 3 looks to the `asm` module structure shown in Figure 2:

Figure 3. Directory structure for the asm-tree module

© Copyright IBM Corporation 1994, 2007. All rights reserved.



The difference, of course, is that the JAR files contain different classes, and the definition in the ivy.xml file in Figure 2 describes the `asm-tree` module. (As it happens, the `asm-tree` module doesn't define any dependencies in its ivy.xml file.)

Growing with Ivy

Now that you have the basics of using Ivy, I'll go over some other useful Ant tasks.

Rendering reports

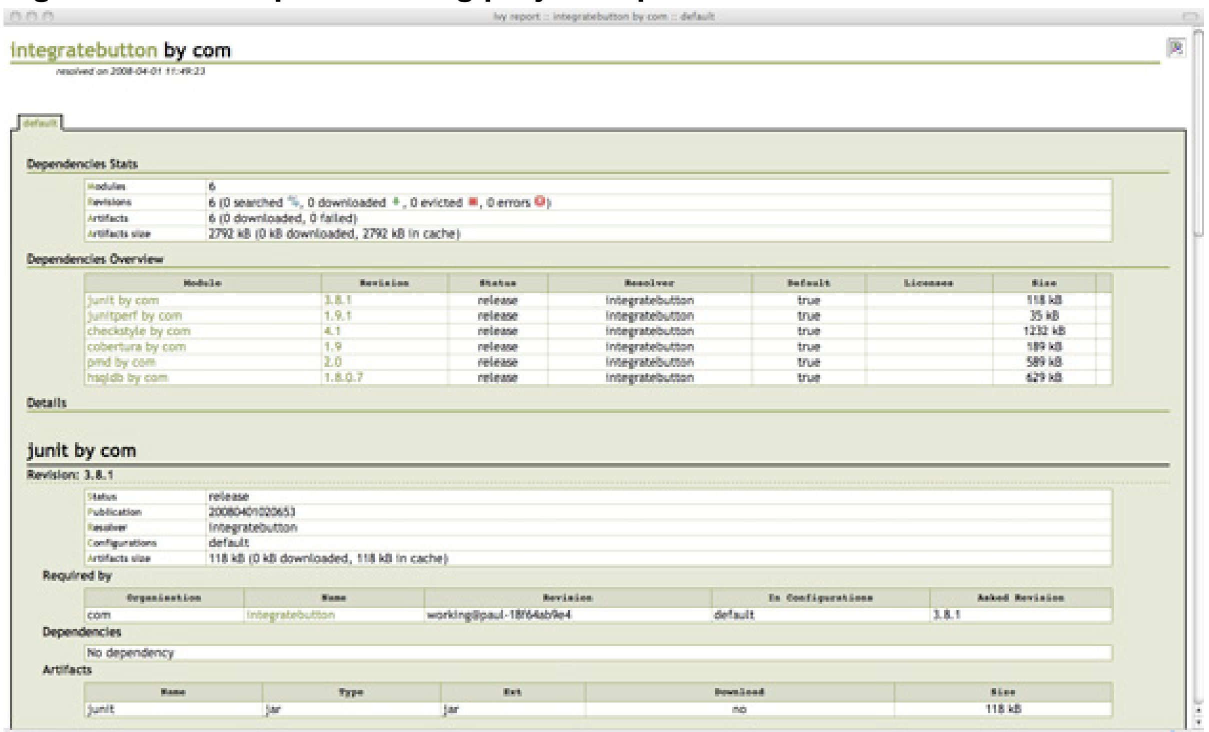
Ivy provides a task for reporting the dependent files in a project. Listing 7 demonstrates calling Ivy's `report` Ant task to create a list of dependencies:

Listing 7. Generating an Ivy dependency report from Ant

```
<target name="ivy-report" depends="init-ivy">
  <ivy:report todir="${target.dir}/reports/ivy"/>
</target>
```

An HTML report generated by the script in Listing 7 displays a list of the project's dependent files. Figure 4 shows the report:

Figure 4. HTML report showing project dependencies



Additional tasks

© Copyright IBM Corporation 1994, 2007. All rights reserved.

Many more Ant tasks for Ivy are available for you to use — from generating a POM file for Maven to cleaning the local file-system cache. Table 1 shows some of Ivy's Ant tasks and their purposes:

Table 1. Other Ivy Ant tasks

Task	Purpose
`settings`	Most useful for authenticating to a host containing a repository
`cachepath`	Overrides the default cache path where downloaded files are hosted on a local file system
`repreport`	Generates reports among several modules in the repository
`install`	Installs a module and all of its dependencies
`makepom`	Creates a pom.xml file from an ivy.xml file for Maven's use
`cleancache`	Cleans the local file-system cache to force a reretrieve of JAR files from repository on next build

See Resources to learn about other Ant tasks available in Ivy.

It all depends

Versioning binaries

Ivy does *not* obviate the need to version-control JAR files. I've often seen that teams provided with an HTTP-accessible repository forget to place the files in a version-control system at all. If you need to recreate the software a year from now and your HTTP repository isn't centrally managed, it could be extremely difficult to do so. Using an HTTP-accessible version control repository such as Subversion makes this less of a burden because you can manage centrally *and* provide HTTP access.

Ivy centralizes dependent files and reduces the bloat that can occur when development teams copy JAR files from one version-control repository to the next. If you're working a simple project, it probably won't slow you down too much to check in JAR files to your version-control system or use some of the other techniques I listed at the start of this article. But as your project gets larger or if you work in an enterprise environment that uses common files, a common approach becomes necessary. In either case, Ivy makes defining project dependencies more consistent and approachable, so it's worth your time to investigate the use of Ivy in your projects.

© Copyright IBM Corporation 1994, 2007. All rights reserved.

Resources

Learn

- Apache Ivy: Visit the Ivy project site for documentation, tutorials, and community resources.
- "Ivy in 4.2 steps" (Andrew Glover, testearly.com, June 2007): Get up and running with Ivy in a few steps.
- Ivy / Maven2 Comparison (Apache Ant Ivy Project): A discussion of the differences between Ivy and Maven 2 dependency management.
- *Ant in Action* (Steven Loughran and Erik Hatcher, Manning, 2007): Chapter 11 in this excellent book is dedicated to dependency management using Ivy.
- *Automation for the people* (Paul Duvall, developerWorks): Read the complete series.
- Browse the technology bookstore for books on these and other technical topics.
- developerWorks Java technology zone: Hundreds of articles about every aspect of Java programming.

Get products and technologies

- Ant: Download Ant and start building software in a predictable and repeatable manner.
- Ivy: Download Ivy.

Discuss

- Improve Your Code Quality discussion forum: Regular developerWorks contributor Andrew Glover brings his considerable expertise as a consultant focused on improving code quality to this moderated discussion forum.
- Accelerate development space: Regular developerWorks contributor Andrew Glover hosts a one stop portal for all things related to developer testing, Continuous Integration, code metrics, and refactoring.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Paul Duvall
Paul Duvall is the CTO of Stelligent Incorporated, an Agile consultancy that helps development teams deliver production-ready software. He is the co-author of the Addison-Wesley Signature Series book *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley Professional, 2007; Jolt Award 2008 winner). He also contributed to the *UML 2 Toolkit* (Wiley, 2003) and the *No Fluff Just Stuff Anthology* (Pragmatic Programmers, 2007).

© Copyright IBM Corporation 1994, 2007. All rights reserved.