

Java Dependency Management with Ivy

Part II

In Part I [1] of Java Dependency Management with Ivy I looked at basic Ivy [2] usage and configuration using Ant [3] and the Ivy Eclipse[4] plugin, IvyDE[5]. I demonstrated how Ivy can be used to download modules (dependencies) from a repository, such as the Maven Repository [6] and cache them locally, negating the need to check them into a source control system.

However there are some scenarios where the Maven repository is not suitable:

1. Your development team may not have direct access to the Maven repository or you want to prevent each module from being downloaded more than once.
2. You may want to restrict or specify the modules your development team has access to.
3. You want to use libraries such Microsoft's SQL Server JDBC driver [7] or your own propriety JARs that are not hosted in the Maven repository.

Ivy and IvyDE can be easily configured to look at custom repositories. In this article I will discuss a way of setting up a local public repository and a shared repository, and how to reference them from Ivy and IvyDE.

In my previous article I also explained the difference between a repository and a cache. As I am going to look at repositories in a little more detail it is worth repeating the distinction. A cache is usually local. When you do a build, Ivy checks the cache to see if you already have the required modules. If you do, it uses them, otherwise it looks in one or more repositories for them and downloads them. Repositories can be local, but tend to be remote on the internet or on a central server in an organisation. Maven is a repository and stores a large number of modules.

Ivy Repositories

As described by the Ivy "Adjusting Default Settings" documentation[8], Ivy uses three kinds of repositories:

- Public - a public repository in which most modules, and especially third party modules, can be found
- Shared - a repository which is shared between all the members of a team and hosts proprietary modules
- Local - a repository which is private to the user and hosts modules specific to them.

The documentation goes into more depth, but basically by default the Public repository is the Maven Repository, the Shared repository is a directory called `shared` in the user's home directory and the Local repository is in a folder called `local` in the user's home

directory. The Shared repository is intended to be shared by the development team and would usually, but not by default, be hosted on an internal server. The local repository is private to each user and hosted on their local machine.

By default, Ivy first checks the Local repository, then the Shared repository and finally the Public repository. This can be changed as described in the documentation. If for some reason you are unable or do not want to access the Maven repository directly a shared repository can be set up to proxy the Maven repository and be used as the public repository. How to set this up is documented on the Ivy website under Building a Repository [9], but the documentation lacks detail, so I'll describe it in more detail here.

Glossary of Terms

Some of the terms used in the Ivy documentation and the various XML files are not as intuitive as they could be. Particularly the difference between a module and an artifact. To try and make this article a little easier to understand I'm including a brief glossary of terms below. More detailed explanations can be found on the Ivy Terminology page [11].

Term	Description
Organisation	An organisation is either a company, an individual, or simply any group of people that produces software. The Ivy "organisation" is very similar to the Maven POM "groupId".
Module	A module is a self-contained, reusable unit of software that, as a whole unit, follows a revision control scheme. Ivy is only concerned about the module deliverables known as artifacts, and the module descriptor that declares them.
Module Descriptor	A module descriptor is a generic way of identifying what describes a module: the identifier (organisation, module name, branch and revision), the published artifacts, possible configurations and their dependencies. The most common module descriptors in Ivy are Ivy Files, xml files with an Ivy specific syntax, and usually called ivy.xml.
Artifact	An artifact is a single file ready for delivery with the publication of a module revision, as a product of development. Compressed package formats are often preferred because they are easier to manage, transfer and store. For the same reasons, only one or a few artifacts per module are commonly used. However, artifacts can be of any file type and any number of them can be declared in a single module.

Revision	<p>A unique revision number or version name is assigned to each delivered unique state of a module. Ivy can help in generating revision numbers for module delivery and publishing revisions to repositories, but other aspects of revision control, especially source versioning, must be managed with a separate version control system.</p> <p>Therefore, to Ivy, a revision always corresponds to a delivered version of a module.</p>
----------	--

Hosting A Repository

A repository can be a few things, including a shared drive or a webserver. Setting up a webserver to host repositories negates the need for sharing drives, which has got to be a good thing as sharing a network drive introduces all sorts of security issues and usually requires all the machines using it to be on the same network, etc. Any webserver that can serve files can be used. My personal preference is for Apache [11]. After installing Apache create a directory called `Ivy` in the `htdocs` directory. This is the location for your repositories and can be accessed as `http://myserver/ivy`.

Creating A Public Repository

A locally hosted public repository holds the required subset of modules from external repositories such as the Maven repository. The `ivy:install` task is used to download the required modules and install them in the local public repository. It requires source and destination resolvers which are usually specified in a settings file:

```
<!-- public-repo-settings.xml -->
<ivysettings>
  <resolvers>
    <ibiblio name="ibiblio" m2compatible="true" />
    <filesystem name="public-repo-resolver">
      <artifact pattern="${dest.dir}/[organisation]/[module]/[types]/[artifact]-[revision].[ext]"/>
    </filesystem>
  </resolvers>
</ivysettings>
```

The `ibiblio` resolver is for the Maven repository. The destination resolver, `public-repo-resolver`, describes the location (`${dest.dir}`), path (`[organisation]/[module]/[types]`) and file format (`[artifact]-[revision].[ext]`) the modules will be written to for the local public repository.

`commons-lang` [12] is a good example of a module in the Maven repository that can be easily installed in a local public repository:

```
<!-- build.xml -->
<project default="create-public-repo" xmlns:ivy="antlib:org.apache.ivy.ant">
  <target name = "create-public-repo" >
    <property name="dest.dir" value="C:/.../htdocs/ivy/public" />
  </target>
</project>
```

```

<ivy:settings id="repo.settings" file="public-repo-settings.xml"/>
<ivy:cleancache/>

<ivy:install      settingsRef="repo.settings"
                  organisation="commons-lang"
                  module="commons-lang"
                  revision="2.0"
                  from="ibiblio"
                  to="dest-resolver"
                  overwrite = "true"
                  haltonfailure = "yes"
                  transitive="true"/>

    <ivy:cleancache/>
  </target>
</project>

```

`dest.dir` specifies the location of the public repository and the `ivy:settings` target specifies the location of the settings file. Creating a repository using `ivy:install` resolves all modules to the local cache. This can cause problems when using the Ivy Ant client or IvyDE later on, so it's best to clean it out before and after the install using the `ivy:cleancache` task. However, if you are trying to resolve a number of dependencies for a module, as you'll see with Hibernate shortly, it is worth commenting `ivy:cleancache` out so that modules are only downloaded once.

The `ivy:install` target is given the settings id so it can find the settings file, the organisation, name and revision of the module and the source and destination resolvers. `overwrite` instructs the task to overwrite any previous installations of the module, `haltonfailure` stops the install process if there is an error and `transitive` tells `ivy:install` to download and install all of the modules dependencies.

Running `build.xml` downloads and caches `commons-lang` locally and then installs it in the public repository. If you look in the `Ivy` directory in Apache's `htdocs` directory you will see a new directory called `public`. Inside `public` you will find the following structure:

```

commons-lang
  commons-lang
    ivys
      ivy-2.0.xml
      ivy-2.0.xml.md5
      ivy-2.0.xml.sha1
    jars
      commons-lang-2.0.jar
      commons-lang-2.0.jar.md5
      commons-lang-2.0.jar.sha1

```

The `ivy-2.0.xml` file describes the module and its artifacts. `commons-lang-2.0.jar` and is obviously the `commons-lang` JAR, the `.md5` files contain a checksum that can be used to verify the modules once they are downloaded and the `.sha1` files contain a hash

function that can be used for secure download.

At this point `commons-lang` is installed in the local public repository and ready to use, but before I describe how to configure the Ivy client to use it, I am going to look at a more complex example.

`commons-lang` is a very simple module without any dependencies. More complex modules, such as Hibernate [13] are dependant on a number of other modules and these should also be installed in the local public repository. Fortunately this is what the `transitive` attribute is for:

```
<ivy:install settingsRef="repo.settings"
  organisation="org.hibernate"
  module="hibernate"
  revision="3.2.5.ga"
  from="ibiblio"
  to="dest-resolver"
  overwrite = "true"
  haltonfailure = "yes"
  transitive="true"/>
```

Unfortunately, due to the badly configured Maven repository some of the modules fail to install with a message suggesting they are not in the repository. This is where the Ivy documentation really falls down as it suggests that you should "...download those artifacts manually, and copy them to your destination repository to complete the installation." You could do that, but there are a couple of other things to try first. The failing modules are:

- `commons-attributes.commons-attributes-compiler-2.1`
- `javax.security.jacc-1.0`
- `javax.transaction.jta-1.0.1B`

`commons-attributes.commons-attributes-compiler-2.1` can be installed, simply by specifying an `ivy:install` task for it:

```
<ivy:install      settingsRef="repo.settings"
  organisation="commons-attributes"
  module="commons-attributes-compiler"
  revision="2.1"
  from="ibiblio"
  to="dest-resolver"
  overwrite = "true"
  haltonfailure = "yes"/>
```

Unfortunately `javax.security.jacc-1.0` and `javax.transaction.jta-1.0.1B` are too badly configured in Maven to be resolved like this, so they have to be downloaded manually, but they can be installed using `ivy:install`, complete with Ivy XML files, rather than just being copied into the public repository.

First the `javax.security.jacc-1.0` and `javax.transaction.jta-1.0.1B` JARs need to be downloaded, renamed as `javax.security.jacc-1.0.jar` and

javax.transaction.jta-1.0.1B.jar respectively and put into a temporary location (e.g. C:\Temp\repo-src). Then a new resolver needs to be added to the Ivy settings file to enable `ivy:install` to locate the JARs:

```
<!-- public-repo-settings.xml -->
<ivysettings>
  <resolvers>
    <ibiblio name="ibiblio" m2compatible="true" />
    <filesystem name="dest-resolver">
      <artifact pattern="\${dest.dir}/[organisation]/[module]/[type)s/[artifact]-[revision].[ext]"/>
    </filesystem>
    <filesystem name="local-resolver">
      <artifact pattern="\${src.dir}/[organisation].[artifact]-[revision].[ext]"/>
    </filesystem>
  </resolvers>
</ivysettings>
```

Then a `src.dir` property needs to be created in `build.xml` and set to `C:/Temp/repo-src` and then `ivy:install` tasks can be created for `javax.security.jacc-1.0` and `javax.transaction.jta-1.0.1B`:

```
<ivy:install      settingsRef="repo.settings"
                 organisation="javax.security"
                 module="jacc"
                 revision="1.0"
                 from="local-resolver"
                 to="dest-resolver"
                 overwrite = "true"
                 haltonfailure = "yes"/>

<ivy:install      settingsRef="repo.settings"
                 organisation="javax.transaction"
                 module="jta"
                 revision="1.0.1B"
                 from="local-resolver"
                 to="dest-resolver"
                 overwrite = "true"
                 haltonfailure = "yes"/>
```

With the `ivy:install` tasks for `commons-attributes.commons-attributes-compiler-2.1`, `javax.security.jacc-1.0` and `javax.transaction.jta-1.0.1B` added along with the task for `org.hibernate.hibernate-3.2.5.ga`, Hibernate can be successfully installed in the public repository by running `build.xml`.

Configuring Ivy to Use a Local Public Repository

To demonstrate the use of the local public repository I am going to use the example from *Java Dependency Management with Ivy – Part I*:

```
import org.apache.commons.lang.WordUtils;

public class IvyAnt
```

```

{
    public static void main(String[] args)
    {
        final String msg = "hello, world!";
        System.out.println(WordUtils.capitalizeFully(msg));
    }
}

```

As you can see this code uses the `WordUtils` class from `commons-lang` to capitalise a string. The Ivy file is shown below and will download `commons-lang` from the Maven repository by default.

```

<!-- ivy.xml -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<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="Purple Tube" module="IvyAnt" status="integration"/>

    <dependencies>
        <dependency org="commons-lang"
            name="commons-lang"
            rev="2.0"
            conf="default"/>
    </dependencies>
</ivy-module>

```

To get Ivy to request `commons-lang` from your local public repository you need to override its default `ivysettings.xml` file and provide your own. The default `ivysettings.xml` file is included in Ivy's JAR file and looks like this:

```

<ivysettings>
    <settings defaultResolver="default"/>
    <include url="${ivy.default.settings.dir}/ivysettings-public.xml"/>
    <include url="${ivy.default.settings.dir}/ivysettings-shared.xml"/>
    <include url="${ivy.default.settings.dir}/ivysettings-local.xml"/>
    <include url="${ivy.default.settings.dir}/ivysettings-main-chain.xml"/>
    <include url="${ivy.default.settings.dir}/ivysettings-default-chain.xml"/>
</ivysettings>

```

Briefly, here the public, shared and local repository resolver configuration files are specified as well as files that describe the order in which they should be used. A more detailed description is provided in the “Adjusting Default Settings” documentation.

To override the `ivysettings.xml` from the Ivy JAR, create a file called `ivysettings.xml` in the same place as `ivy.xml` and copy the xml above into it. The settings file will be automatically picked up by the Ivy Ant task, but you'll need to tell IvyDE about it by:

1. Right clicking on “ivy.xml [*]” and selecting properties.
2. Going to the Main tab and ticking “Enable project specific settings”.
3. Entering the path to `project:///ivy-settings.xml` into the Ivy settings path.
4. Clicking Ok.

Next you need a file telling Ivy how to find dependencies in the local public repository:

```
<!-- ivysettings-purple-public.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ivysettings>
<ivysettings>
  <resolvers>
    <url name="public">
      <artifact pattern="http://myserver/ivy/public/[organisation]/[module]/[type]s/[artifact]-[revision].[ext]" />
      <ivy pattern="http://myserver/ivy/public/[organisation]/[module]/ivys/ivy-[revision].xml"/>
    </url>
  </resolvers>
</ivysettings>
```

The above XML specifies a new url resolver that can be added to Ivy's settings. The artifact tag tells Ivy where to look for the dependency (`http://myserver/ivy/public`) and what format the name of the JAR and the Ivy file will be in. In this case `[organisation]` relates to the `org` attribute of the Ivy dependency XML tag, `[artifact]` and `[module]` relate to the `name` attribute, `[revision]` relates to the `rev` attribute and `[ext]` defaults to `jar`.

The above XML needs to go into another file called something like `ivy-purple-settings.xml` and can then either go together with `ivy.xml` and `ivysettings.xml` or, more sensibly I think, into the `ivy` directory in the Apache `htdocs` directory of the repository. Either way you need to modify `ivysettings.xml` so Ivy can find it:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ivysettings>
<ivysettings>
  <settings defaultResolver="default"/>
  <include url="ivysettings-purple-public.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-shared.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-local.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-main-chain.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-default-chain.xml"/>
</ivysettings>
```

or

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ivysettings>
<ivysettings>
  <settings defaultResolver="default"/>
  <include url="http://localhost/ivy/ivysettings-purple-public.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-shared.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-local.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-main-chain.xml"/>
  <include url="{ivy.default.settings.dir}/ivysettings-default-chain.xml"/>
</ivysettings>
```

With these files in place Ivy and IvyDE will now be able to find `commons-lang` in the local

public repository and download it to the cache.

Setting up a Shared Repository

There is no need to set up a shared repository unless you have something to put in it. You could, if you wanted to, use your public repository as the shared repository. However, I think it is good to keep third party and proprietary modules separate. You may even want to keep them on separate servers, or create a shared repository and continue to use the Maven repository as your public repository. Ivy gives you lots of options!

The Microsoft SQL Server JDBC driver is a good example of a library that is not hosted in the Maven Repository and should therefore be put in a shared repository. At the time of writing Hibernate 3.3.2 is another good example. However, writing client code to demonstrate the use of either is quite verbose, so I am going to use a custom JAR instead.

My custom JAR, `net.purpletube.goodmusic-0.1.jar` contains a single class:

```
package net.purpletube.goodmusic;

import org.apache.commons.lang.WordUtils;

public class FavoriteAlbum
{
    public String getTitle()
    {
        return "ROMULOUS";
    }

    public String getArtist()
    {
        return WordUtils.capitalizeFully("EX DEO");
    }
}
```

that is also dependent on the `WordUtils` class from the Apache `commons-lang` library. Therefore any client that uses the `FavoriteAlbum` class will have dependencies on both `net.purpletube.goodmusic-0.1.jar` and `commons-lang.jar`. `net.purpletube.goodmusic-0.1.jar` is not, of course, in the Maven Repository or the local public repository, but `commons-lang.jar` is. Therefore `net.purpletube.goodmusic-0.1.jar` should be published to a shared repository, with a dependency on `commons-lang.jar`, but `commons-lang.jar` should not.

Before `net.purpletube.goodmusic-0.1.jar` can be published, it needs an Ivy file to describe it and its dependent modules :

```
<!-- net.purpletube.goodmusic-0.1.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<ivy-module version="2.0">
    <info organisation="net.purpletube" module="goodmusic" revision="0.1"/>
    <publications>
        <artifact name="goodmusic" type="jar" ext="jar" conf="default"/>
    </publications>
</ivy-module>
```

```

</publications>
<dependencies>
  <dependency org="commons-lang"
              name="commons-lang"
              rev="2.0"
              conf="default"/>
</dependencies>
</ivy-module>

```

This XML specifies the organisation, name and version of the module, the artifacts (JARs in this case) to be published and the dependency on `commons-lang`.

`ivy:install` and `ivy:publish` can both be used to create a shared repository. `ivy:install` is easier to use, so I'll describe it here. As with installing the local public repository, you need a source resolver and a destination resolver in another settings file:

```

<!-- ivysettings-repo.xml -->
<ivysettings>
  <resolvers>
    <filesystem name="local-resolver">
      <artifact pattern="${src.dir}/[organisation].[artifact]-[revision].
[ext]"/>
      <ivy pattern="${src.dir}/[organisation].[module]-[revision].xml"/>
    </filesystem>
    <filesystem name="dest-resolver">
      <artifact pattern="${dest.dir}/[organisation]/[module]/[type]s/[artifact]-
[revision].[ext]"/>
    </filesystem>
  </resolvers>
</ivysettings>

```

As with the local public repository, `local-resolver` describes the file name format (`[organisation].[artifact]-[revision].[ext]`) and location (`${src.dir}`) of modules to be installed. The `dest-resolver` describes the location (`${dest.dir}`), path (`[organisation]/[module]/[type]s`) and file format (`[artifact]-[revision].[ext]`) the modules will be written to.

The install task is configured as follows:

```

<target name = "create-shared-repo" >
  <property name="dest.dir" value="C:/.../htdocs/ivy/shared" />
  <property name="src.dir" value="C:/Temp/repo-src" />

  <ivy:settings id="repo.settings" file="shared-repo-settings.xml"/>
  <ivy:cleancache/>

  <ivy:install
      settingsRef="repo.settings"
      organisation="net.purpletube"
      module="goodmusic"
      revision="0.1"
      from="local-resolver"
      to="dest-resolver"
      overwrite = "true"
      haltonfailure = "yes"/>

```

```
<ivy:cleancache/>
</target>
```

As with the public repository `ivy:install` task, `src.dir` specifies the location of the dependency to install to the shared repository, `dest.dir` specifies the location of the shared repository and the `ivy:settings` target specifies the location of the settings file. The `ivy:install` target is given the settings id so it can find the settings file, the organisation, module and revision of the dependency and the source and destination resolvers. `overwrite` instructs the task to overwrite any previous installations of the dependency and `haltonfailure` stops the install process if there is an error. You'll notice that `transitive` is missing, this is because we don't want `common-lang` installed to the shared repository.

To install `net.purpletube.goodmusic-0.1.jar`, simply copy it and `net.purpletube.goodmusic-0.1.xml` to `C:\Temp\repo-src` and run the Ant script. If you look in the `Ivy` directory in Apache's `htdocs` directory you will see a new directory called `shared`. Inside there you will find the following structure, just like in the public repository:

```
net.purpletube
  goodmusic
    ivys
      ivy-0.1.xml
      ivy-0.1.xml.md5
      ivy-0.1.xml.sha1
    jars
      goodmusic-0.1.jar
      goodmusic-0.1.jar.md5
      goodmusic-0.1.jar.sha1
```

The `ivy-0.1.xml` file describes the dependency. `goodmusic-0.1.jar` is obviously the `net.purpletube.goodmusic` JAR, the `.md5` files contain a checksum that can be used to verify the module once it is downloaded and the `.sha1` files contain a hash function that can be used for secure download.

The shared repository is now set up, installed and ready to use.

Configuring Ivy to Use a Shared Repository

To demonstrate the use of the shared repository we need a piece of client code that uses the `FavoriteAlbum` class:

```
import net.purpletube.goodmusic.FavoriteAlbum;

public class GoodMusicClient
{
    public static void main(String[] args)
    {
```

```

    final FavoriteAlbum fav = new FavoriteAlbum();

    final StringBuilder buf = new StringBuilder(fav.getTitle());
    buf.append(" by ");
    buf.append(fav.getArtist());

    System.out.println(buf);
}
}

```

and the appropriate Ivy file:

```

<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="purpletube.net" module="IvyAnt" status="integration"/>
  <dependencies>
    <dependency org="net.purpletube"
      name="goodmusic"
      rev="0.1"
      conf="default"/>
  </dependencies>
</ivy-module>

```

Note that only `net.purpletube.goodmusic-0.1.jar` is specified. To get `commons-lang-2.0.jar`, we're relying on the fact that the shared repository knows that `net.purpletube.goodmusic.jar` depends on it.

As with the local public repository you need a file telling Ivy how to find dependencies in the shared repository:

```

<!-- shared-repo-settings.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ivysettings>
<ivysettings>
  <resolvers>
    <url name="shared">
      <artifact pattern="http://myserver/ivy/shared/[organisation]/[module]/[type]s/[artifact]-[revision].[ext]" />
      <ivy pattern="http://myserver/ivy/shared/[organisation]/[module]/ivys/ivy-[revision].xml"/>
    </url>
  </resolvers>
</ivysettings>

```

Put this alongside `public-repo-settings.xml` in the `ivy` directory in the Apache `htdocs` directory. Then modify the local `ivysettings.xml` file to use the `public-repo-settings.xml` rather than the default shared repository:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ivysettings>
<ivysettings>
  <settings defaultResolver="default"/>
  <include url="http://myserver/ivy/ivysettings-purple-public.xml"/>

```

```
<include url="http://myserver/ivy/ivysettings-purple-shared.xml"/>
<include url="{ivy.default.settings.dir}/ivysettings-local.xml"/>
<include url="{ivy.default.settings.dir}/ivysettings-main-chain.xml"/>
<include url="{ivy.default.settings.dir}/ivysettings-default-chain.xml"/>
</ivysettings>
```

With these files in place Ivy and IvyDE will now be able to find `net.purpletube.goodmusic-0.1.jar` in the shared repository and download it to the cache. It will also see that `net.purpletube.goodmusic-0.1.jar` depends on `commons-lang` and that it is not in the shared repository, so will download it from the local public repository.

Running `GoodMusicClient` will write “ROMULOUS by Ex Deo”^[14] to the console, proving that, although not straightforward, creating Ivy repositories is quite easy to do.

Sidebar

The first part of Java Dependency Management with Ivy was easy to write as using the Ivy clients is easy. The only complication I had was a new version of Ivy and IvyDe coming out while I was writing it! Creating a repository and getting to grips with the terminology and Ivy's poor documentation is quite demanding. It's not that Ivy doesn't have documentation, in fact it has quite a lot, it's just that it's so badly written with lots of assumption and very little detail.

This is the third version of this article. The first version described how to set up a very simple shared repository, without using `ivy:install` and then I intended to adjust it for dependencies of the modules installed in it. That turned out to be confusing, so I adjusted it to use `ivy:install` and handled the dependencies of the modules from the start.

Then I sent it over to Mr Jez Higgins for a read and he pointed out that he would like to know how to set up an alternative public repository. Due to bugs in the Maven Repository, configuring this is not straight forward. The Ivy documentation touches on this, but doesn't really address it properly or give a proper solution. So in this third version I look at creating an alternative public repository that doubles as a shared repository for your own modules.

Acknowledgements

Thank you to Shawn Castrianni, Geoff Clitheroe, Joshua Tharp and Tom Widmer of the Ivy Users list for helping me when I could not see the wood for the trees, and Jez Higgins and Steve Love for review and direction.

References

[1] Java Dependency Management with Ivy – Part1

[2] Ivy, The Agile Dependency Manager: <http://ant.apache.org/ivy/>

[3] Ant: <http://ant.apache.org/>

[4] Eclipse IDE: <http://www.eclipse.org/>

[5] Ivy Eclipse Plugin: <http://ant.apache.org/ivy/ivyde/>

[6] Maven Repository: <http://mvnrepository.com/>

[7] Microsoft SQL Server JDBC Driver: <http://msdn.microsoft.com/en-us/data/aa937724.aspx>

[8] Adjusting Default Settings: <http://ant.apache.org/ivy/history/2.1.0-rc1/tutorial/defaultconf.html>

[9] Building a Repository: <http://ant.apache.org/ivy/history/latest-milestone/tutorial/build-repository.html>

[10] Ivy Terminology: <http://ant.apache.org/ivy/history/latest-milestone/terminology.html>

[11] Apache Webserver: <http://httpd.apache.org/>

[12] Apache commons-dbc Library: <http://commons.apache.org/lang/>

[13] Hibernate: <https://www.hibernate.org/>

[14] Ex Deo: <http://www.myspace.com/exdeo>