Ivy - Beginners Guide

Maithilish
maithilish@gmail.com

Contents

Chapter 1 Introduction

In Java projects it is always convenient to reuse the popular libraries like Apache Commons, Log4j etc. rather than develop your own libraries from scratch. In simple projects, that depends on couple of libraries, dependency can be managed by adding these jars to the project manually. But as project becomes more sophisticated they may end up with dozens of external jars. When project is developed by a team then members may add these jars to project independently and may lead to broken builds. In all these situations dependency management becomes quite cumbersome.

Apache Maven is most popular build manager for Java projects which combines build manager and dependency manager. Maven also comes with other features like project information to handle the complete build process of Java projects. But for some reason you don't want all these features or prefer Ant for build process then Ivy would be the next best choice for automated dependency management.

Ivy is a popular dependency manager from Apache. It fits perfectly with Ant to automate dependency management and improves the project build system. IvyDE is a Eclipse plugin which integrates Ivy dependency management with Eclipse.

Manual v/s Automated Dependency Management

For fairly long time I use to maintain a lib directory where all important libraries like apache-commons, jdbc drivers, dom4j, log4j etc with corresponding source, javadoc jars are placed. For each project in Eclipse jars are added manually to project's build path. There was also additional efforts of linking respective source and javadoc to the library so that they are available in context help. Later while distributing the application same set of jar file paths are to be added to Ant build file. Developer has to be careful to include proper version of these libraries to avoid runtime errors in production. No doubt it is lot of unnecessary work.

In a recent project, list of dependencies grew beyond a dozen. It was high time to move to automated dependency management. With Ivy things become extremely simple. All that was required was to add a ivy.xml file to project and build path. This simple file enumerates the dependencies and version. IvyDE will take care all linkages like source and javadoc etc. Personal lib directory is no longer necessary. Ivy will fetch the required dependencies from a public repository and places them in its cache for further use. When a new library is required for project just add a dependency line to ivy.xml and Ivy will take care to rest of the chore.

About this Guide

Ivy is a powerful piece of software and highly configurable. Ivy site has some excellent tutorials and detailed reference documentation. But a beginner may soon get lost by multiple configurable options and struggle to get it right in first try.

This step-by-step guide explains how to set up Ivy, use Ivy to automate dependency management, set up enterprise repository and finally integrate Ivy with Eclipse through IvyDE.

Basic objective of this guide it to provide clear understanding of Ivy to the beginners. As such we have used minimum required configuration to get the task done in all examples. Once readers are comfortable with basic concepts, they can explore the advanced features of Ivy to suit their requirement.

Part I Ivy

This part covers

- installation of Ivy
- basic tasks like Resolve and Retrieve
- core concepts like pattern, resolvers etc.
- setting up of local and shared repository
- publish your own modules to repository

Chapter 2 Installation

We require Apache-ant and Apache-ivy to work through this part of book. In case Ant is already installed in your system ignore the ant installation commands. We are installing ant at /opt/ant, but any other location is fine. Adjust settings accordingly. Download latest Ivy and Ant from Apache site.

```
tar -C /opt/ant/ -xzvf apache-ant-1.8.2-bin.tar.gz
tar -xzvf apache-ivy-2.2.0-bin-with-deps.tar.gz
cp apache-ivy-2.2.0/ivy-2.2.0.jar /opt/ant/apache-ant-1.8.2/lib
export ANT_HOME=/opt/ant/apache-ant-1.8.2
export PATH=$PATH:$ANT_HOME/bin
```

ANT_HOME and PATH has to be set to run ant. Exports may be moved to .bash_profile in Linux so that they are always set.

Test the installation

Throughout this guide a work directory like $HOME/work is used to run examples. But it can be anything. For readability and consistency we will be using phrases like create a file in work dir, add a ivy.xml to workdir and run from work dir etc. which simply means work from some directory. For each exercise start with a clean directory to avoid any errors and ensuing confusion.

Now to test the installation add following build.xml file to work dir

build.xml

```
<project name="test ivy" default="test"
                         xmlns:ivy="antlib:org.apache.ivy.ant">

       <target name="test" description="Test ivy installation">
              <ivy:settings />
       </target>
</project>
```

and run ant

```
ant
```

Successful build indicates that ant and ivy installation is fine. Typical error would be

build.xml:5: Problem: failed to create task or type antlib:org.apache.ivy.ant:settings

this may be due to

- environment variable ANT_HOME not set properly
- ivy.jar is missing from ANT_HOME/lib

Chapter 3 Ivy Terminology

Brief description of some of terms used by Ivy are

Term	Description
Organisation	Name of company, individual or team which created the software. Example: org.apache etc.
Module	Module is a self-contained, reusable unit of software. Module typically has version. Example: commons-lang, log4j etc.
Module Descriptor	Xml file containing the description or metadata of a module. Usually this file is named as ivy.xml
Artifact	Artifact is a single file ready for distribution. In java it is typically a jar file. But it may be of any file type like zip, gz etc. Module contains one or more artifacts. Jar, Source jar and Javadoc jar are examples of artifact.
Type	Type denotes category of artifact. Example: jar, src, source, doc, bundle etc.
Artifact file name Extension	Extension of an artifact. Example: jar, zip, tar, tar.gz etc
Module Revision	unique revision number or version name of particular release of module.
Status of Revision	indicates how stable a module revision is. Ivy defines following status • integration: continuous build, a nightly build etc. • milestone: distributed but not yet finished fully. • release: tested and completed.
Repository	Distribution site where ivy can find modules, descriptors and artifacts. Repository can be public, local or shared.
Ivy Settings	Ivy works without any specific configuration through default settings. But default settings can be overridden through xml based settings file. Usually this file is named as ivysettings.xml

Ivy Tasks

Ivy comes with it own set of tasks which can be called from Ant build file. This guide covers following Ivy Tasks

Task Name	Description
Resolve	Resolves the dependencies described in ivy.xml and places the resolved dependencies in ivy cache.
Cachepath	Constructs an ant path consisting of artifacts in ivy cache which can be referred in other ant tasks through Ant path mechanism.
Retrieve	Copies the resolved dependencies from cache to a specified directory
Install	Installs a module to a specified repository. In this guide Install task is used to install libraries from public repository to a local/shared repository
Publish	Publish a module to a repository.

Chapter 4 Resolve Task

Resolve task resolve dependencies described in ivy.xml and put the resolved dependencies in the ivy cache.

Create a package in work dir

```
mkdir -p src/in/ex/ivy
```

Add following java file to this dir.

Example.java

```
package in.ex.ivy;

import org.apache.commons.lang.StringUtils;

public class Example {
        public static void main(String[] args) {
                String string = StringUtils.upperCase("Ivy Beginner Guide");
                System.out.println(string);
        }
}
```

This simple programme depends on commons-lang module from Apache.

Instead of adding commons-lang package to build path manually, let us delegate the dependency management to Ivy.

Add a file named ivy.xml to work dir.

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="in.ex" module="ivy-example" status="integration">
   </info>

   <dependencies>
        <dependency org="commons-lang" name="commons-lang" rev="2.6" />
   </dependencies>
</ivy-module>
```

ivy.xml provides details about our project and dependencies. Ivy will read this file and manage the dependencies.

This file has 3 sections

1. ivy-module : this is standard root element with version,schema etc.

2. info : this section is information about our project.
   - organisation : organisation or company.
   - module : name of the project
   - status : release status – milestone, integration or release.

3. dependencies : one or more dependencies of this project.
    - dependency :
        - org : organisation which provides the module
        - name : module name
        - rev : revision or version of the module

We are indicating that our project is dependent on commons-lang revision 2.6

Add ant build.xml to work dir.

build.xml

```
<project name="ivy example" default="resolve"
        xmlns:ivy="antlib:org.apache.ivy.ant">

      <target name="resolve" description="resolve dependencies with ivy">
            <ivy:resolve />
      </target>
</project>
```

In project element we have added ivy namespace so that we can use ivy tasks. <ivy:resolve /> in target named resolve is the resolve task. This will resolve the dependencies described in ivy.xml and places the resolved dependencies in ivy cache.

Now run ant. Ivy goes into job of resolving the dependencies we mentioned in ivy.xml. If you are connected to internet Ivy will try to fetch the commons-lang revision 2.6 from a public repository.

Ivy fetches the commons-lang from net and places them in cache.

Console output shows that resolve step has gone through successfully. Information about the Resolve is provided in last portion of output

```
[ivy:resolve] :: resolution report :: resolve 26675ms :: artifacts dl 745018ms
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   1   |   1   |   1   |   0   ||   3   |   3   |
        ---------------------------------------------------------------------
```

- conf — ivy is using default configuration
- module — One module is resolved. As Ivy has downloaded this module dwnlded is one.
- artifacts — there are 3 artifacts i.e. jar, source and javadoc in this module. As Ivy has downloaded them dwnlded is three.

In this example we have not told Ivy where to look for modules. Ivy uses concept called Resolvers which defines location of a repository. Ivy has used the default resolvers. Default public resolver points to http://repo1.maven.org/maven2. Ivy has fetched the module from this repository.

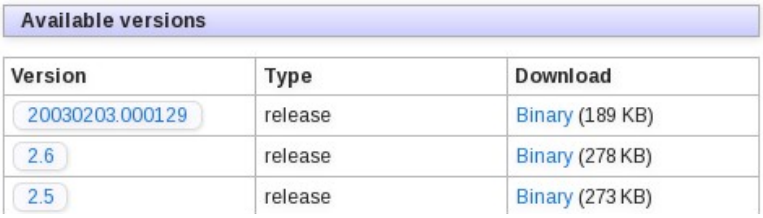
In case a module itself has other dependencies then Ivy will fetch all required dependencies. These are known as transitive dependencies. For example log4j is depends on javax.mail, junit, jermanio.spec and oro. Dependency line <dependency org="log4j" name="log4j" rev="1.2.16"/> will download log4j plus four dependent modules. For some reason you don't want to download dependent module then add

transitive=”false” to dependency line.

Dependency line

The dependency line <dependency org="commons-lang" name="commons-lang" rev="2.6" /> in ivy.xml requires couple of explanations.

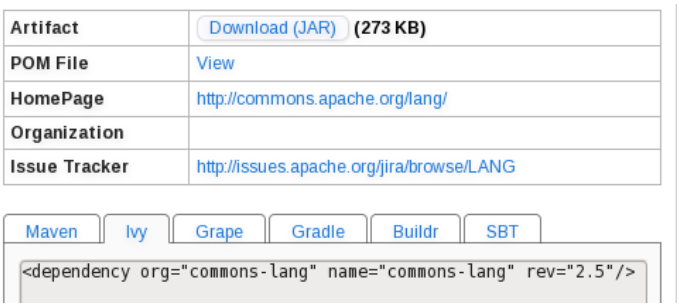
How to construct this line. Visit http://mvnrepository.com/ and search for module. For example a search for commons-lang will yield all matching commons module. Select commons-lang among them. Version available in the repository will be displayed



Available versions

Version	Type	Download
20030203.000129	release	Binary (189 KB)
2.6	release	Binary (278 KB)
2.5	release	Binary (273 KB)

Click on 2.5 or whatever version you are interested in to get the dependency line for Ivy among other build systems like maven, grape etc. Select Ivy tab and there you will have the exact dependency line as required for Ivy. Cut and paste this line to ivy.xml



Artifact	Download (JAR) (273 KB)
POM File	View
HomePage	http://commons.apache.org/lang/
Organization	
Issue Tracker	http://issues.apache.org/jira/browse/LANG

Maven | Ivy | Grape | Gradle | Buildr | SBT

```
<dependency org="commons-lang" name="commons-lang" rev="2.5"/>
```

Another thing about this particular dependency line is that organisation is mentioned as commons-lang where as it should have been apache.org. For some reasons while uploading the module to maven repository it was mentioned as commons-lang instead of apache.org and it is continuing likewise. Rather than making assumption about the org or name it is better to search for the module in maven repository and copy the dependency line as provided there.

Ivy Cache

On first run resolve will fetch the artifacts from a repository and place them in cache. Subsequent call to resolve either from this project or another project for the resolved artifact, resolve task finishes without any delay. These is because Ivy finds dependencies in cache.

By default ivy cache is at $HOME/.ivy2. Beyond that we need not bother much about the cache.

Chapter 5 Cachepath Task

Constructs an ant path consisting of artifacts in ivy cache.

In last chapter we used Resolve task to resolve and put the artifacts into cache. Now to compile the program we require classpath reference to these artifacts. Cachepath task constructs the ant path for this.

Change build.xml as follows

build.xml

```
<project name="ivy example" default="resolve"
        xmlns:ivy="antlib:org.apache.ivy.ant">

       <target name="resolve" description="resolve dependencies with ivy">
              <ivy:resolve />
                     <ivy:cachepath pathid="default.classpath"/>
       </target>

             <target name="compile" depends="resolve" description="Compile">
              <mkdir dir="build/classes" />
              <javac srcdir="src" destdir="build/classes">
                    <classpath refid="default.classpath" />
              </javac>
       </target>
</project>
```

After resolve ivy constructs Ant path named default.classpath. This path points to resolved artifacts in cache. Later in javac task we have referred it with refid. In this example javac will use the resolved artifacts in cache to compile the Example.java

In case ivy:cachepath is called directly without a ivy:resolve then resolve is internally called before constructing the path.

Note – It is to be noted that artifacts are not copied from cache either by Resolve or by Cachepath task. Cachepath constructs a path points to artifacts in cache.

Chapter 6 Retrieve Task

Instead of cachepath explained in last chapter, better approach is to copy the dependencies to project space and use standard ant path creation.

Retrieve task copies resolved dependencies to a specified location in filesystem. Once dependencies are properly copied to project workspace use standard ant path creation to build the project.

Change the build.xml as follows

build.xml

```
<project name="ivy example" default="resolve"
        xmlns:ivy="antlib:org.apache.ivy.ant">

       <target name="resolve" description="Resolve and Retrieve with ivy">
            <ivy:resolve />
             <ivy:retrieve sync="true" type="jar" />
       </target>
</project>
```

After resolve ivy will copy the resolved artifacts from cache to a newly created lib directory.

In case ivy:retrieve is called directly without a ivy:resolve then resolve is internally called before retrieving the artifacts.

Important attributes that are used in this task are

Sync set to true will ensure that any extra files in lib directory is deleted.
Type set to jar tells ivy to copy only jar artifacts. Source and javadoc artifacts are ignored.

Use pattern attribute to change location where retrieved files are to be placed

```
<ivy:retrieve sync="true" type="jar" pattern="myfolder/[artifact]-[revision].[ext]" />
```

This will retrieve the artifacts to myfolder directory. Other aspects of the pattern will be explained later.

Some modules like log4j use ***bundle*** as type for jar artifact. Use following line to handle that variation

```
<ivy:retrieve sync="true" type="jar,bundle" />
```

Chapter 7 Ivy Core Concepts

Repositories

Till now we have used public repository to resolve dependencies. But for many reasons you may want to set up your own repository

- internet access – companies may have internet access or usage policy. This may limit or even restrict access to internet. Setting up a enterprise repository helps to overcome this issue.
- reliability/accuracy – modules available in public repository may have quality issues.
- security – module available in public repository may have security issues and Information Security policy of the company may deny the access to these repository.

Default configuration of Ivy allows following types of repository

- Public – repositories available in internet
- Local – private repository, access is restricted to the user
- Shared – a common repository shared between the members of a team

Settings Files

Till now we have used default settings of Ivy to carry out tasks. Settings files are xml files usually called ivysettings.xml and they are used to override default settings.

Default settings are defined by Ivy in its own ivysettings.xml which is packaged in ivy.jar.

Patterns

Pattern is used by Ivy in many tasks and settings. Ivy places the artifacts in the filesystem but it doesn't stipulates a particular directory structure. Pattern is used to define a directory structure or the artifact file name. Ivy uses patterns to

- to name artifacts
- to place artifacts in proper directory
- to access or search artifacts in repositories

Pattern is composed of tokens. These tokens are replaced by actual values to evaluate a particular module or artifact. Token are surrounded by square brackets as [ext].

Important tokens

- [organisation] organisation name
- [module] module name
- [revision] revision name
- [artifact] artifact name (or id)
- [type] artifact type
- [ext] artifact file extension
- [conf] configuration name

We can mix tokens, Ivy variables and actual directory names to compose a pattern.

For example we want to install or add module developed by xyz.com to our local repository. Module Pigo has three artifacts - class jar, source jar and javadoc jar. Values for token will be

Token	Value
[organisation]	com.xyz
[module]	pigo
[revision]	1.1
[artifact]	pigoapp
[type]	jar – for class jar file source – for the source jar file doc – for javadoc jar file
[ext]	jar
[conf]	default – we are using the default configuration of the Ivy

For token [type] value changes depending on the artifact and for all other tokens values are same for all three artifacts.

Now we can use patterns to control the directory structure and artifact naming as

- `[organisation]/[module]/[type]s/[artifact]-[revision].[ext]`

  This pattern makes Ivy to

  - create a directory with organisation name
  - under that create a directory with module name
  - under that create three directories for 3 types – jar, source and doc. As there is a **s** after [type] actual directory names will be jars, sources and docs
  - then place the artifacts under respective directories. File name will have artifact name and revision

  Result would be

  com.xyz/pigo/jars/pigoapp-1.1.jar
  com.xyz/pigo/sources/pigoapp-1.1.jar
  com.xyz/pigo/docs/pigoapp-1.1.jar

- `[organisation]/[module]/[artifact]-[type]-[revision].[ext]`

  This pattern makes Ivy to

  - create a directory with organisation name
  - under that create a directory with module name
  - under that place all three artifacts. File name will have artifact name, type and revision

Result would be

com.xyz/pigo/pigoapp-jar-1.1.jar
com.xyz/pigo/pigoapp-source-1.1.jar
com.xyz/pigo/pigoapp-doc-1.1.jar

Later while resolving the dependencies Ivy will use this pattern to handle directories and file appropriately.

- In retrieve task in last chapter we had given pattern as

```
myfolder/[artifact]-[revision].[ext]
```

This pattern makes Ivy to

- to retrieve required artifacts and name them with artifact name, revision and ext.
- then copy the artifact to myfolder

Result would be

myfolder/pigoapp-1.1.jar
myfolder/pigoapp-1.1.jar
myfolder/pigo/pigoapp-1.1.jar

Resolvers

One of the configurable items in ivysettings.xml is Resolvers.

Repositories are not homogeneous. It may be hosted on a web server, local filesystem, vfs filesystem, ssh server etc. Layout of the repositories and naming of artifacts differs among repositories.

Ivy uses concept called Resolver to contact repository and fetch the files. For this Ivy has to

- decide network or filesystem protocol to access the repository.
- then get hold of ivy file which may follow different layout/naming in different repositories.
- then find proper artifacts which may follow different layout/naming in different repositories.

Resolver defines these aspects so that Ivy can properly resolve the dependencies.

Resolvers defines a list of dependency resolvers usable by Ivy. Each dependency resolver is identified by its name.

Ivy ships with built-in dependency resolvers that handle most common needs. There are two types of resolvers

- standard – these resolvers are used by Ivy for actual resolve task
- composite – these resolvers delegate the work to standard resolvers

List of important Built-in Resolvers:

Name	Type	Description
IBiblio	Standard	Finds artifacts on ibiblio.
FileSystem	Standard	This very performant resolver finds ivy files and artifacts in your file system.
Url	Standard	Finds ivy files and artifacts in any repository accessible with urls.
Chain	Composite	Delegates the finding to a chain of sub resolvers.
Dual	Composite	Delegates the finding of ivy files to one resolver and of artifacts to another.

ivysettings.xml with a url resolver :

ivysettings.xml

```
<ivysettings>
  <resolvers>
    <url name="url-example">
       <ivy pattern="http://ivyrep.xyz.com/[module]/[revision]/ivy-[revision].xml" />
       <artifact pattern="http://ivyrep.xyz.com/[module]/[revision]/[artifact].[ext]" />
    </url>
  </resolvers>
</ivysettings>
```

This will look for ivy.xml and artifacts based on ivy pattern and artifact pattern.

Examples

Defines a resolver called xyz using the maven 2 public repository to find module metadata (using maven 2 poms) and artifacts.

```
<resolvers>
        <ibiblio name="xyz" m2compatible="true" />
<resolvers>
```

Defines a resolver named test using url. This is equivalent to the ibiblio resolver mentioned above.

```
<resolvers>
       <url name="test" m2compatible="true">
           <artifact pattern=
              "http://repo1.maven.org/maven2/[organisation]/[module]/[revision]/
[artifact]-[revision].[ext]"/>
       </url>
<resolvers>
```

Defines a resolver using ibiblio pointing to http://repo.pentaho.org/artifactory/repo instead of default http://repo1.maven.org/maven2

```
<resolvers>
    <ibiblio name="pentaho" m2compatible="true"
             root="http://repo.pentaho.org/artifactory/repo" />
<resolvers>
```

Defines a local filesystem resolver named mylocal. Variable ivy.default.ivy.user.dir by default points .ivy2 directory in user home directory and we are using local directory under that. Repository layout is as per the rest of pattern [organisation]/[module]/[type]s/[artifact]-[revision]-[type]s.[ext]

```
<resolvers>
   <filesystem name="mylocal">
     <ivy pattern=
         "${ivy.default.ivy.user.dir}/local/[organisation]/[module]/[type]s/[artifact]-
[revision]-[type]s.[ext]" />
     <artifact pattern=
         "${ivy.default.ivy.user.dir}/local/[organisation]/[module]/[type]s/[artifact]-
[revision]-[type]s.[ext]" />
   </filesystem>
</resolvers>
```

Defines a Chain resolver. Chain is a composite resolver which delegates to a chain of sub resolvers. With this Ivy will try to resolve through local filesystem resolver and on failure it will try maven2 public repository.

```
<settings defaultResolver="chain-resolver" />
<resolvers>
   <chain name="chain-resolver">
     <filesystem name="mylocal">
         <ivy pattern=
           "${ivy.default.ivy.user.dir}/local/[organisation]/[module]/type]s/artifact]-
[revision]-[type]s.[ext]" />
         <artifact pattern=
           "${ivy.default.ivy.user.dir}/local/[organisation]/[module]/type]s/artifact]-
[revision]-[type]s.[ext]" />
     </filesystem>
     <ibiblio name="ibiblio" m2compatible="true" />
  </chain>
</resolvers>
```

Default Resolvers

Ivy ships with a set of resolvers which will be used in absence of our own resolvers.

Name	Type	Description
local	standard	Filesystem resolver points to $HOME/.ivy2/local
shared	standard	Filesystem resolver points to $HOME/.ivy2/shared
public	standard	Ibiblio resolver points http://repo1.maven.org/maven2
main	composite	Chain and Dual resolver to shared and public
default	composite	Chain resolver to local and main

Hierarchical Relationship is as follows

- default
    - local
    - main
        - shared
        - public

Ivy by default uses resolver named default which first try local and on failure try with shared and finally public.

We can check this by running resolve task after deleting the .ivy2/cache directory and disabling internet connection of the machine. Following output shows that it has tried local, shared and public in that order

Now we have gone through some important concepts in Ivy and with these we can start building our own repositories.

Chapter 8 Install Task

Resolves a module and its dependencies from a repository and install them in another repository. Install copies module's ivy file and artifacts from source repository to a target repository.

To build private repository like local or shared repository we have to copy the module's files from public repository to our private repository.

Local Repository

Built in Variable ${ivy.default.ivy.user.dir} points to .ivy2 directory in user home. By default, the local repository location is ${ivy.default.ivy.user.dir}/local.

Add following files to work dir.

build.xml

```
<project name="localrepository" default="install"
         xmlns:ivy="antlib:org.apache.ivy.ant">

      <target name="install" description="--> install modules to localreporsitory">
               <ivy:install organisation="commons-lang" module="commons-lang"
                     revision="2.6" transitive="true"  overwrite="false"
                    from="public" to="local"/>
    </target>
</project>
```

This will install commons-lang 2.6 from public repository to local filesystem repository.

Run ant and install task will create a new local directory in $HOME/.ivy2.

List of files in $HOME/.ivy2/local

```
[m@m install4]$ find /home/m/.ivy2/local -type f -print
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/javadocs/commons-lang.jar
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/javadocs/commons-lang.jar.md5
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/javadocs/commons-lang.jar.sha1
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/jars/commons-lang.jar
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/jars/commons-lang.jar.md5
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/jars/commons-lang.jar.sha1
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/sources/commons-lang.jar
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/sources/commons-lang.jar.md5
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/sources/commons-lang.jar.sha1
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/ivys/ivy.xml
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/ivys/ivy.xml.md5
/home/m/.ivy2/local/commons-lang/commons-lang/2.6/ivys/ivy.xml.sha1
```

We have not provided any definition for local, public resolver and pattern to be followed by Ivy while installing the artifacts in local repository. How come Ivy is able to resolve these aspects. In the output of build you will have following line

[ivy:resolve] :: loading settings :: url = jar:file:/opt/ant/apache-ant-1.8.2/lib/ivy-.2.0.jar!
/org/apache/ivy/core/settings/ivysettings.xml

This gives hint that ivy is using a default ivysettings.xml packaged in ivy-2.0.jar. Extract this archive to get ivysettings.xml in org/apache/ivy/core/settings with following contents

/org/apache/ivy/core/settings/ivysettings.xml

```
<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>
```

By default ivy comes with shared, default, local, public, main resolvers. For each of these it points to separate settings file. In same directory we will get these five settings files.

/org/apache/ivy/core/settings/ivysettings-public.xml

```
<ivysettings>
        <resolvers>
                <ibiblio name="public" m2compatible="true"/>
        </resolvers>
</ivysettings>
```

/org/apache/ivy/core/settings/ivysettings-local.xml

```
<ivysettings>
   <property name="ivy.local.default.root"
               value="${ivy.default.ivy.user.dir}/local" override="false"/>
   <property name="ivy.local.default.ivy.pattern"
               value="[organisation]/[module]/[revision]/[type]s/[artifact].[ext]"
                   override="false"/>
   <property name="ivy.local.default.artifact.pattern"
              value="[organisation]/[module]/[revision]/[type]s/[artifact].[ext]"
                   override="false"/>
   <resolvers>
   <filesystem name="local">
   <ivy pattern="${ivy.local.default.root}/${ivy.local.default.ivy.pattern}" />
   <artifact pattern="${ivy.local.default.root}/$ivy.local.default.artifact.pattern}"/>
   </filesystem>
   </resolvers>
</ivysettings>
```

From these files ivy gets definition for public and local resolvers.

Ivy pattern and artifact pattern are `[organisation]/[module]/[revision]/[type]s/[artifact].[ext]`. Local repository will be of this layout.

We have now successfully created a local repository. It is always a good idea to do a resolve and retrieve the modules to ensure that all artifacts are retrieve from our repository without any naming conflicts.

To do that add ivy.xml to install directory

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="in.ex" module="simpleivy" status="integration"></info>
   <dependencies>
        <dependency org="commons-lang" name="commons-lang" rev="2.6" />
   </dependencies>
</ivy-module>
```

This is the same file which we had used earlier in resolve task.

Add following snippet to build.xml and run ant resolve

```
<target name="resolve" description="resolve dependencies with ivy">
      <ivy:retrieve />
</target>
```

Now Ivy will do a resolve and retrieve and place the retrieved artifacts in lib directory.

```
[m@m install1]$ ls -lrt lib
total 2236
-rw-rw-r-- 1 m m  372982 2011-01-17 03:51 commons-lang-2.6-sources.jar
-rw-rw-r-- 1 m m 1624033 2011-01-17 03:51 commons-lang-2.6-javadoc.jar
-rw-rw-r-- 1 m m  284220 2011-01-17 03:51 commons-lang-2.6.jar
```

As we have not provided retrieve pattern in retrieve task Ivy as used default retrieve pattern.

Now local repository is good to resolve. But if you observe the output you will have line as follows

[ivy:resolve] confs: [default]
[ivy:resolve] found commons-lang#commons-lang;2.6 in public
[ivy:resolve] :: resolution report :: resolve 172ms :: artifacts dl 25ms

Output indicates that commons-lang is found in public. Default set up is Ivy has first look in local, then in shared and then public. But Ivy is still resolving from public repository even though module is in local.

Reason is cache. As explained earlier ivy will check the cache and resolve. During install ivy has copied the module to cache and then to local. That copy is still in cache. During resolve it will get this copy and says that it has found it in public.

Use cleancache ant task to clear the cache. Add following snippet to build.xml and run ant clean-cache

```
<target name="clean-cache" description="--> clean the cache">
        <ivy:cleancache />
</target>
```

This task will delete the cache directory in .ivy2. With this cache is cleared.

Now run resolve task and ivy will resolve from local repository. Output indicates that commons-lang is found in local and triggers a download to cache.

Overriding the default settings

There are two ways to override default settings either through setting property in ant target or through ivysettings.xml file

As an example let us change the layout of local repository. Change build.xml as

build.xml

```
<project name="shared repository" default="install"
         xmlns:ivy="antlib:org.apache.ivy.ant">

    <target name="install" description="local at non default location" >

      <property name="ivy.local.default.ivy.pattern"
           value="[organisation]/[module]/ivys/ivy-[revision].xml" />
      <property name="ivy.local.default.artifact.pattern"
           value="[organisation]/[module]/[type]s/[artifact]-[revision].[ext]" />
        <ivy:install organisation="commons-lang" module="commons-lang" revision="2.6"
                transitive="true" overwrite="false"
                from="public" to="local"/>
    </target>
</project>
```

We have used ant property to change ivy pattern and artifact pattern. Install still uses default local resolver but layout of repository is changed as per new pattern.

Alternatively we can use ivysettings.xml to override default settings. Following ivysettings.xml defines resolver named myresolver. It points to $HOME/.ivy2/local but layout pattern is different from default local resolver. Place this file along with build.xml and Ivy will load this file instead of default file that comes with Ivy. Ivy will get only three resolvers myresolver, mylocal and mypublic. Default resolver default, main, local, shared and public will not be available as default ivysettings.xml is not loaded. Use mylocal and mypublic in install task from and to attributes.

ivysettings.xml

```
<ivysettings>
 <property name="ivy.local.default.root"
           value="${ivy.default.ivy.user.dir}/local" override="false" />
 <property name="ivy.local.default.ivy.pattern"
           value="[organisation]/[module]/ivys/ivy-[revision].xml" override="false" />
 <property name="ivy.local.default.artifact.pattern"
           value="[organisation]/[module]/[type]s/[artifact]-[revision].[ext]"
           override="false" />

 <settings defaultResolver="myresolver"/>
   <resolvers>
     <chain name="myresolver">
      <filesystem name="mylocal">
        <ivy pattern="${ivy.local.default.root}/${ivy.local.default.ivy.pattern}" />
       <artifact pattern=
             "${ivy.local.default.root}/${ivy.local.default.artifact.pattern}" />
      </filesystem>
      <ibiblio name="mypublic" m2compatible="true" />
     </chain>
   </resolvers>
</ivysettings>
```

In ivysettings.xml we have defined two resolvers

- mylocal – filesystem based repository with root at .ivy2/local as defined by variable ivy.local.default.root
- mypublic – default maven2 public repository

Ivy pattern `[organisation]/[module]/ivys/ivy-[revision].xml` has placed ivy.xml in ivys directory. Artifact pattern `[organisation]/[module]/[type]s/[artifact]-[revision].[ext]` has placed jar, source and javadoc artifacts in jars, sources and javadocs directory. Even though all three artifacts are named as commons-lang-2.6.jar Ivy will distinguishes them by the directory name.

Shared repository

Only difference between local and shared repository is that all users can access a shared repository and whereas local repository is private to a user.

Default configuration comes with a resolver named shared and it point to $HOME/.ivy2/shared directory. But in case you don't want a shared repository in your home directory then shared repository can be created in other locations as follows.

Shared repository should be in a directory to which all users have read and write access. Let us create a shared repository at /opt/ivy/repository/shared

```
mkdir -p /opt/ivy/repository/shared
chown -R ivy.ivy /opt/ivy
chmod -R 777 /opt/ivy
```

This will create a directory accessible by all users.

Use following variables to override the default values of shared resolver.

ivy.shared.default.root
ivy.shared.default.ivy.pattern
ivy.shared.default.artifact.pattern

build.xml

```
<project name="shared repository" default="install"
         xmlns:ivy="antlib:org.apache.ivy.ant">

    <target name="install" description="--> install modules to shared reporsitory" >

        <property name="ivy.shared.default.root" value="/opt/ivy/repository/shared"/>
        <ivy:install organisation="commons-lang" module="commons-lang" revision="2.6"
                 transitive="true" overwrite="false"
                 from="public" to="shared"/>
    </target>
</project>
```

Now ivy.shared.default.root variable is pointed to /opt/ivy/repository/shared. With this all users can read and write shared repository.

To access this shared repository users have to override ivy.shared.default.root variable before calling ivy:resolve in build.xml

Moving Repository to Web Server

Shared repository explained in last section can be accessed by all users but they have to be on same machine

or access the server through telnet. In enterprise set up, developers work from desktop and repository will be on a server. Following method may be used to set up repository accessible via network.

Shared repository has to be on server which has web server running on it and maintained by a administrator.

Create a directory for the repository under Document Root folder of the httpd.

```
mkdir /var/www/html/ivyrepo
chown apache.apache /var/www/html/ivyrepo
chmod 775 /var/www/html/ivyrepo
usermod -aG apache m                # m is username
service httpd start                 # start the apache web server
```

As example we are using Apache http server with its default Document root at /var/www/html. Users who are allowed to install modules are added to apache group. These users have to use following build.xml to install the modules.

build.xml

```
<project name="shared repository" default="install"
         xmlns:ivy="antlib:org.apache.ivy.ant">

    <target name="install" description="--> install modules to shared reporsitory" >

        <property name="ivy.shared.default.root" value="/var/www/html/ivyrepo"/>
        <ivy:install organisation="commons-lang" module="commons-lang" revision="2.6"
                 transitive="true" overwrite="false"
                 from="public" to="shared"/>
    </target>
</project>
```

Developers have to use following ivysettings.xml to access the repository from their desktops.

ivysettings.xml

```
<ivysettings>
        <property name="web.ivy.pattern"
                value="[organisation]/[module]/[revision]/[type]s/[artifact].[ext]"
                override="false" />
        <property name="web.artifact.pattern"
                value="[organisation]/[module]/[revision]/[type]s/[artifact].[ext]"
                override="false" />
        <settings defaultResolver="chain" />
        <resolvers>
          <chain name="chain">
             <url name="web">
                <ivy pattern="http://localhost/ivyrepo/${web.ivy.pattern}" />
                <artifact pattern="http://localhost/ivyrepo/${web.artifact.pattern}" />
             </url>
             <ibiblio name="public" m2compatible="true" />
          </chain>
        </resolvers>
</ivysettings>
```

Here we are using URL resolver. Change http://localhost in ivy and artifact pattern to URL of the server.

Repository Manager

Web based Repository manager like Archiva, Artifactory or Nexus may be used to maintain the repository. These front ends has a security model for the repository access and group of users can upload the modules to repository through them.

Chapter 9 Publish task

Publish task publishes module's artifacts and resolved descriptor to a repository. This task is used to publish the modules developed by us or modules which are not available in other repositories.

This task will not create the artifacts. Artifacts are created separately by the build system.

To publish a module we require artifacts and descriptor file ivy.xml. In actual project we use ant to build the project and create jar for module, source and javadoc etc.

For demonstration of publish task, instead of actual artifact, let use zero byte files as three dummy artifacts

```
touch simpleivy-jar.jar  simpleivy-javadoc.jar  simpleivy-source.jar
```

This will create three zero byte files.

Now add ivy.xml

ivy.xml

```
<ivy-module version="2.0">
  <info organisation="com.xyz" module="simpleivy" />
    <publications>
        <artifact name="simpleivy" type="jar" ext="jar"/>
        <artifact name="simpleivy" type="javadoc" ext="jar"/>
        <artifact name="simpleivy" type="source" ext="jar"/>
    </publications>

  <dependencies>
        <dependency org="commons-lang" name="commons-lang" rev="2.6" />
   </dependencies>

</ivy-module>
```

This will be the descriptor for our module simpleivy. We have indicated in publications element that this publication has three artifacts with their types and ext.

Now add build.xml with publish task

build.xml

```
<project name="localrepository" default="publish"
              xmlns:ivy="antlib:org.apache.ivy.ant">

    <target name="publish" description="Publish this build into repository">
            <ivy:resolve/>
            <ivy:publish pubrevision="1.0" status="release" resolver="local"
                  overwrite="true" >
                  <artifacts pattern="[artifact]-[type].[ext]"/>
            </ivy:publish>
    </target>
</project>
```

In publish task we have given

- pubrevision – publication revision. 1.0
- status – Status of revision. Ivy allows integration, milestone and release as status
- resolver – we want to publish the module to local repository
- artifacts pattern – this pattern will be used by ivy to search for artifacts. Our artifacts has simple pattern - [artifact]-[type].[ext] like simpleivy-javadoc.jar. This pattern is used only to search for artifacts defined in ivy.xml. It is important to note that while placing the artifacts in repository Ivy will use the pattern provided in resolver definition for layout and name the artifacts.

Now we are ready to publish our simpleivy module to local repository. Run ant and it will publish our modules to local repository as seen from following output.

```
[ivy:publish] :: publishing :: com.xyz#simpleivy
[ivy:publish]   published simpleivy to /home/m/.ivy2/local/com.xyz/simpleivy/1.0/jars/simpleivy.jar
[ivy:publish]   published simpleivy to /home/m/.ivy2/local/com.xyz/simpleivy/1.0/javadocs/simpleivy.jar
[ivy:publish]   published simpleivy to /home/m/.ivy2/local/com.xyz/simpleivy/1.0/sources/simpleivy.jar
[ivy:publish]   published ivy to /home/m/.ivy2/local/com.xyz/simpleivy/1.0/ivys/ivy.xml
```

Ivy has used pattern given default local resolver definition in /org/apache/ivy/core/settings/ivysettings-local.xml to place the files in repository.

- `[organisation]/[module]/[revision]/[type]s/[artifact].[ext]` to place ivy.xml
- `[organisation]/[module]/[revision]/[type]s/[artifact].[ext]` to place artifacts

One more thing to observe is that now you will have two ivy file in your work directory.

- Orginal ivy.xml which you had given
- Second ivy-ivy.xml generated by publish

This second ivy-ivy.xml file is resolved descriptor which is also known as delivered ivy file. When we invoked publish task first it run deliver task. Deliver task generates a resolved descriptor of the module, based upon the last resolve done. The resolved ivy file contains updated information about the delivered module, such as revision and status. And this second file is actually published to repository.

Part II IvyDE

This part cover

- installation of Ivy and IvyDE plugins
- build a eclipse project with Ivy
- explore IvyDE features

Chapter 10 Installation

Update Site

Select Eclipse Install New Software option and add, workwith following update site

http://www.apache.org/dist/ant/ivyde/updatesite/

It will display plugins available for installation

- Apache Ivy library
- Apache IvyDE Eclipse plugin

select both and proceed to install. This will install ivy, ivy ant tasks and ivyide

Manual installation

This method involves installation of IvyDE and Ivy plugins.

- IvyDE Plugin

This plugin links core Ivy to Eclipse. Download the IvyDE plugin from http://ant.apache.org/ivy/ivyde/download.cgi. For Eclipse Indigo download apache-ivyde-2.1.0.201008101807-RELEASE.tar.gz and uncompress to get two folders containing the artifacts to deploy in your Eclipse. Copy them to eclipse installation directory as follows:

plugins/org.apache.ivyde.eclipse_2.1.0.201008101807-RELEASE.jar to $ECLIPSE_HOME/plugins
features/org.apache.ivyde.feature_2.1.0.201008101807-RELEASE.jar to $ECLIPSE_HOME/features

- Ivy Plugin

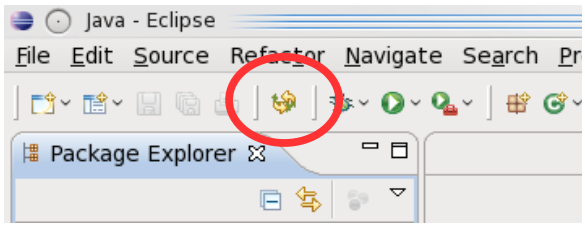
This plugin is Ivy and Ant task. Get the latest Ivy feature and plugin from following sites

http://www.gtlib.gatech.edu/pub/apache//ant/ivyde/updatesite/plugins
http://www.gtlib.gatech.edu/pub/apache//ant/ivyde/updatesite/features

For Eclipse Indigo download and copy them to your eclipse installation directory as follows

org.apache.ivy_2.2.0.final_20100923230623.jar to $ECLIPSE_HOME/plugin
org.apache.ivy.eclipse.ant_2.2.0.final_20100923230623.jar to $ECLIPSE_HOME/plugin
org.apache.ivy.feature_2.2.0.final_20100923230623.jar to $ECLIPSE_HOME/features

After installation restart Eclipse and menu bar will have Resolve button of Ivy



Chapter 11 Resolve

Lets use IvyDE to resolve dependencies of a Java Project.

Create a regular Java Project in Eclipse named simpleivy. Add class named SimpleIvy

in.ex.ivy.SimpleIvy.java

```
package in.ex.ivy;

import org.apache.commons.lang.StringUtils;

public class SimpleIvy {

	public static void main(String[] args) {
		String string = StringUtils.upperCase("Ivy with Eclipse");
		System.out.println(string);
	}
}
```

As expected Eclipse will show errors since Eclipse is unable to find commons.lang in project's build path.

Instead of manually adding commons-lang package to build path let us delegate the dependency management to Ivy. For this first we have to add a file named ivy.xml under the root folder folder of the project.

Ivy.xml

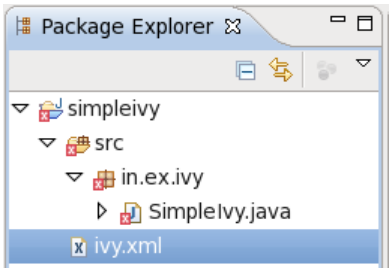
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ivy-module version="2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">

   <info organisation="in.ex" module="simpleivy" status="integration">
   </info>

   <dependencies>
        <dependency org="commons-lang" name="commons-lang" rev="2.6" />
   </dependencies>
</ivy-module>
```

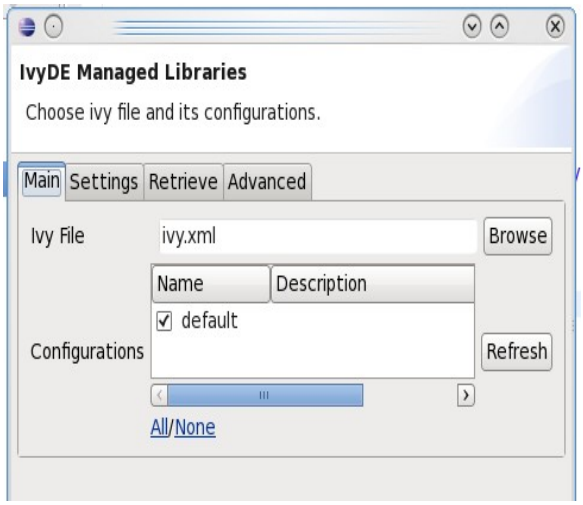
This is the same ivy.xml we had used in ant task earlier. Dependencies element indicates that our project is dependent on commons-lang revision 2.6. We can also use File->New->Other->IvyDE->Ivy File Wizard to add ivy.xml file.

Now the project structure looks as



Next step is add this ivy.xml to build path of the project. For this right click on ivy.xml and choose Add Ivy

Library... from the context menu. IvyDE Managed Libraries window is displayed



In in Main tab, Ivy File is set as ivy.xml which we added to root folder of the project. Leave the Configurations to default.

For the moment lets not bother about other tabs - Settings, Retrieve and Advanced which we shall use in later chapters.

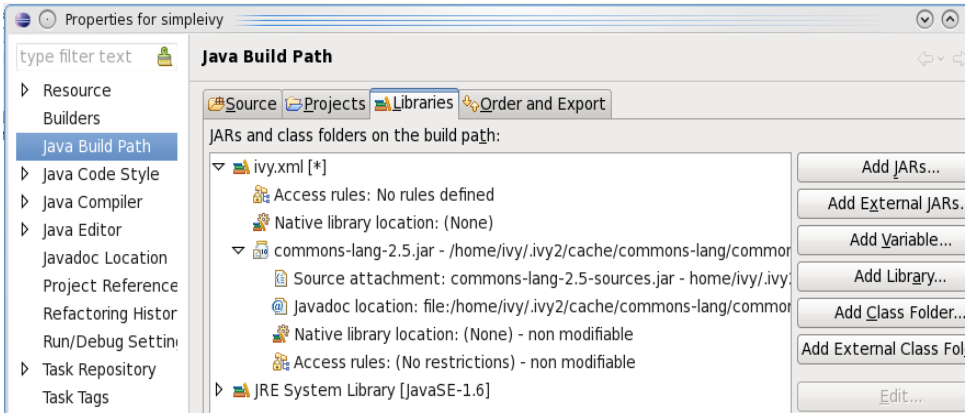
Click on Finish. This will add libraries to Eclipse build path. But these libraries are special in the sense that they are managed by IvyDE. Internally IvyDE will use Ivy to get list dependencies mentioned in ivy.xml and add them to Eclipse Build Path

As soon as you click the Finish button Ivy goes into job of resolving the dependencies you mentioned in ivy.xml. It will check whether Rev 2.6 is in cache else it will fetch from public. Progress Bar indicates the progress of Ivy resolve job

Errors in the project are removed once resolve is over as commons-lang is added to build path of project by IvyDE.

We can add more dependency lines in ivy.xml depending on projects requirement. Ivy will fetch and IvyDE will add them to the project build path.

Let us explore the project build path to know what IvyDE has done. Right click on the project and choose Build Path -> Configure Build Path to Java Build Path window. Select Libraries tab and it will show two libraries ivy.xml and JRE System Library. Expand ivy.xml entry and then commons-lang-2.6.jar to get the library details



As we can see IvyDE has linked source and javadoc automatically. All dependencies mentioned in ivy.xml will be added under ivy.xml [*]

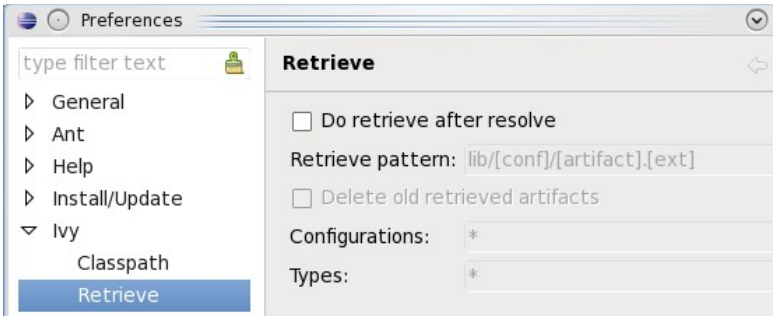
Chapter 12 Retrieve

Retrieve will copy the dependencies to the project directory. This is done after Ivy completes the Resolve step.

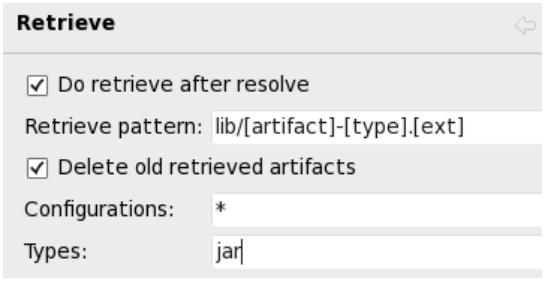
In IvyDE we can enable retrieve at two level either at workspace level or at project level.

Enable Retrieve for all projects in Workspace

Go to Window -> Preferences -> Ivy -> Retrieve



Make changes as follows and save settings



Here we are indicating that

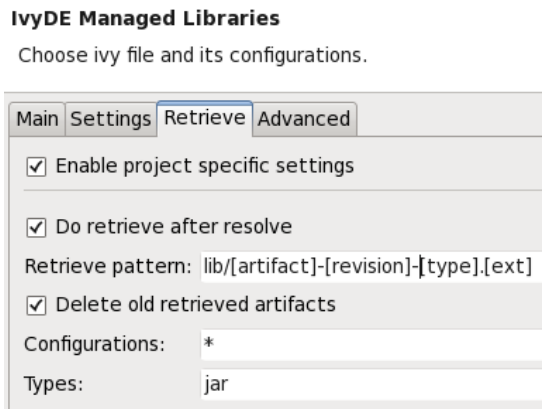
- IvyDE has to retrieve after resolve.
- Retrieve pattern lib/[artifact]-[type].[ext] indicates that retrieved modules are to be placed in lib directory
- IvyDE has to delete stale artifacts which are no longer in dependencies list and lying in lib directory.
- Types - jar means we want only actual library jar files and not javadoc, source jar etc. Multiple comma separated types are allowed.

Refresh the project with F5. Lib directory is created in root folder of the project and dependent jar commons-lang-jar.jar is copied to lib/default directory.

Enable Retrieve at project level

Workspace level Retrieve settings which we had enabled in previous section can be overridden at project level.

Right click on project's ivy.xml and choose Add Ivy Library... in the context menu to get Ivy Managed Libraries window. Select Retrieve tab.



Make changes as detailed here. Here we have added [revision] to retrieve pattern. Save, Resolve and Refresh. In lib/default directory commons-lang-jar.jar is now changed as commons-lang-2.6-jar.jar as we changed the retrieve pattern for this project.

This setting will be specific to this project alone. Other project will take workspace level settings.

Chapter 13 Other Tabs of IvyDE Managed Libraries

Settings Tab

In chapter 8 we had defined a resolver named web through ivysettings.xml to access the repository through web server. We can override the default settings with our own ivysettings.xml in this tab.

This can be either done at workspace level or at project level as explained in last chapter. Place the ivysettings.xml in root folder of workspace for workspace level or in root folder of project to override at project level.

Advanced Tab

This tab is used to

- define accepted types
- define source types and suffixes
- defines javadoc types and suffixes
- change ordering of classpath entries

Again these things can be set at workspace level or at project level.

Chapter 14 Going further

Now that we have covered essential of Ivy, readers should be able to work with Ivy and IvyDE effectively. But it doesn't end there. Ivy comes comes with many advanced features to meet the rigorous needs of an enterprise dependency management system. Below are some of the pointers where readers can gain further insight into features of Ivy to help them to convert their build system into a professional one.

Tutorials

Official documentation is in doc directory of the ivy distribution. It contains set of tutorials which explains some of the features of Ivy. These tutorials walk through the examples that provided in src/example directory of distribution

Ivy Module Configuration

Throughout this guide we used default conf. Ivy provides a concept called Module Configurations. It is a powerful feature in Ivy which groups a set of artifacts and gives meaning to it. Tutorial – Using Ivy Module Configurations explains this concept of Ivy.

Reference Documentation

Official documentation also contains detailed Reference documentation which covers following areas of Ivy

- Setting Files – defines the structure of ivysettings.xml and tags that are allowed in this files
- Ivy Files - defines the structure of ivy.xml and tags that are allowed in this files
- Ant Tasks – provides reference about the various Ivy Ant tasks.