Common Java Libraries

The shared libraries are broken into three categories, as described in the following sections. If you add any jar files to lams_build/lib directory, please update this page.

If you add any libraries, please see the section "Adding a Library" for the list of things that you need to do.

All libraries that will be deployed must be compatible with the GPL license under which LAMS is released. It may be possible to use a non-GPL compatible library for development only, but this makes source code release more complicated and should be avoided if possible. For this reason, j2ee.jar is not included in the build project.

- Using the Shared Libraries
- LAMS Core/Services JAR Files
- Deployed Third Party JAR Files
- Build Related JAR Files
- Ant Build Scripts
- Using the Libraries (Runtime)
- Adding A Libary
- Why do it this way?

Shared Libraries

The shared libraries used in LAMS are stored in the lams_build project, in the lib directory. Other projects must reference the copies stored in lams_build rather than includeing them in the project. This will ensure we are using a consistent set version of libraries.

The core jar/war files should be added to a project as required. For example, the IMS Content Package tool uses the content repository, so it references the lams_content_repository.jar file, but it does not reference lams_learning, lams_authoring, etc as these jars are not required.

The 3rd party jars should be included using the supplied user library, lams_build/3rdParty.userlibraries. This will reference all of the third party jars in one reference. This is designed to make development easier.

A user library is imported (in Eclipse) by:

Bring up the properties for your project.

- Select Java Build Path and then the Libraries tab. Click Add Library.
- On the "Add Library" dialog box, select "User Library" and click "Next".
- If this is the first time, then no user libraries will be listed. Click "Import" and then enter the path of the library definition file and then click "Okay".
- If the user libraries has been set up previously, then you can either select the user library from the list to reuse it, or re-import it using Import.
- Once it is imported into Eclipse, the same user library can be used in many projects. Unfortunately, When the import file is updated, the user library will need to be re-imported. So if the user library is updated, you will need to notify other developers that the need to update their projects.

If you add an entry in the user library, it must start with "\lams_build\lib\". This tells Eclipse to start at the workspace level. Using "..\lams_build\lib\" will not work.

LAMS Core/Services JAR Files

The core jar and service jar files are stored in the common project so that we don't end up with a huge graph of dependent projects (which will require many projects to be open at a time).

When a developer checks in changes to the source code in one of the core or service modules (including lams_common), the developer should update the jar files in lib/lams directory.

The master build script (build.xml in lams_build) will continue to compile the core code from the individual projects, rather than use the cached jars.

There will be difficulties with new versions of code breaking other modules. We just have to work with the problems and rely on developers to work together to keep components in sync as much as possible.

Eventually we will set up an automated build / test based on the common libraries. This should identify problems quickly so that we can deal with them straight away.

Deployed Third Party JAR Files

The following libraries are in the EAR file which makes up the LAMS application. these products, then you should use these versions to avoid any version problems.

Logging: LAMS uses log4j for logging. Commons-logging is included in the ear file to support other libraries.

Folder	Library	Version	License	Vendor	Description
autopatch	tk-autopatch-1.2.0-b2-cvs.jar	1.2.0-b2	Tacit Knowledge License 1.0	Tacit Knowledge	Database migration
	tk-discovery-1.0.0.jar	1.0.0	Tacit Knowledge License 1.0	Tacit Knowledge	
axis	axis-ant.jar				
	axis.jar				
	jaxrpc.jar				
	saaj.jar				
	wsdl4j-1.5.1.jar	1.5.1		IBM	JWSDL
cglib	cglib-nodep-2.1_2.jar	2.1_2	Apache License 2.0		(old one - to be deleted soon!)
	cglib_jboss404GA.jar	4.0.4.GA	Apache License 2.0	JBoss Inc.	JBoss
concurrent	concurrent-1.3.4.jar	1.3.4			
dom4j	dom4j-1.5.2.jar	1.5.2		MetaStuff Ltd.	dom4j : XML framework for Java
fckeditor	FCKeditor-2.3.jar	2.3			
hibernate	hibernate3.jar	3.2.0.cr2		hibernate.org	JBoss
j2ee	jstl.jar	1.1.2		Sun Microsystems, Inc.	JavaServer Pages Standard Tag Library (JSTL)
jakarta-commons	commons-beanutils.jar	1.6	Apache License 2.0	Apache Software Foundation	Jakarta Commons Beanutils
	commons-codec-1.3.jar	1.3	Apache License 2.0	Apache Software Foundation	Jakarta Commons Codec
	commons-collections.jar	3.1	Apache License 2.0	Apache Software Foundation	Commons Collections
	commons-digester.jar	1.6	Apache License 2.0	Apache Software Foundation	Jakarta Commons Digester
	commons-discovery-0.2.jar	0.2	Apache Software License 1.1	Apache Software Foundation	
	commons-fileupload.jar	1.0	Apache Software License 1.1	Apache Software Foundation	File upload component for Java servlets
	commons-io-1.4.jar	1.4	Apache License 2.0	Apache Software Foundation	Jakarta Commons IO
	commons-lang-2.0.jar	2.0	Apache Software License 1.1	Apache Software Foundation	Jakarta Commons Lang
	commons-logging.jar	1.0.4	Apache License 2.0	Apache Software Foundation	
	commons-validator.jar	1.1.4	Apache License 2.0	The Apache Software Foundation	Commons Validator
jakarta-poi	poi-2.5.1-final-20040804.jar	2.5.1-final-20040804			
jakarta-taglibs	standard.jar	1.1.2		Apache Software Foundation	JavaServer Pages Standard Tag Library (JSTL)
jboss	jboss-cache.jar	4.0.2		JBoss Inc.	JBoss
	jboss-common.jar	4.0.2		JBoss Inc.	JBoss
	jboss-jmx.jar	4.0.2		JBoss Inc.	JBoss
	jboss-system.jar	4.0.2		JBoss Inc.	JBoss

```
                 jgroups.jar                          4.0.2
JBoss Inc.                    JBoss

jdom             jdom.jar

jfreechart       gnujaxp.jar                                                GPL
Free Software Foundation      GNU JAXP
                 jcommon-1.0.0.jar                    1.0.0
                 jfreechart-1.0.1.jar                 1.0.1

log4j            log4j-1.2.13.jar                     1.2.13

mysql            mysql-connector-java-5.0.8-bin.jar   5.0.8-bin             GPL (with exception)

odmg             odmg-3.0.jar                         3.0

quartz           quartz.jar                           1.5.2
OpenSymphony                  Quartz Enterprise Job Scheduler

smack            smack.jar
                 smackx.jar

spring           spring.jar
1.2.8                                                                             Spring Framework

struts           antlr.jar
                 jakarta-oro.jar                                            Apache Software License 1.1
                 struts-el.jar
                 struts.jar                           1.2.7                 Apache License 2.0
The Apache Software Foundation   Struts Framework

wddx             wddx.jar
1.1                                                                               WDDX

xstream          jmock-2004-03-19.jar
                 joda-time-0.98.jar                   0.98                  Joda Software License 1.0
Joda.org                      Joda Time
                 stax-1.1.1-dev.jar                   1.1.1-
dev                                                                        StAX is the reference
implementation of the St...
                 stax-api-1.0.jar                     1.0
                 xml-writer-0.2.jar                   0.2
                 xom-1.0b3.jar                        1.0b3
                 xpp3-1.1.3.4d_b4_min.jar             1.1.3.4d_b4_min
                 xstream-1.1.jar                      1.1
```

Notes:

- gnujaxp.jar: we should be able to remove gnujaxp.jar as it is only needed for jfreechart when using Java versions JDK 1.3 OR JDK 1.2.2. This needs to be tested.
- antlr.jar: Supplied with Struts 1.2.4. Appears to be from http://www.antlr.org/. License depends on the version, which is unknown.

Build Related JAR Files

The following jar files are included in the common library as we use them for code generation, build or testing within the IDE.

Build Related Libraries

Folder	Library	Version	License	Vendor	Description
catalina	catalina.jar	4.0.2		JBoss Inc.	JBoss
ehcache	ehcache-1.1.jar	1.2	Apache Software License 1.1		
httpunit	httpunit.jar	1.6			
	js.jar				
	nekohtml.jar		Apache Software License 1.0		
	Tidy.jar				
	xercesImpl.jar				
	xmlParserAPIs.jar				
junit	junit.jar				
middlegen	middlegen-2.1.jar	2.1			
	middlegen-hibernate-plugin-2.1.jar	2.1			
	velocity-1.4-dev.jar	1.4-dev			
strutstestcase	strutstest-2.1.3.jar	2.1.3			
xdoclet	maven-xdoclet-plugin-1.2.3.jar	1.2.3			
	xdoclet-1.2.3.jar	1.2.3			
	xdoclet-apache-module-1.2.3.jar	1.2.3			
	xdoclet-bea-module-1.2.3.jar	1.2.3			
	xdoclet-borland-module-1.2.3.jar	1.2.3			
	xdoclet-caucho-module-1.2.3.jar	1.2.3			
	xdoclet-de-locale-1.2.3.jar	1.2.3			
	xdoclet-ejb-module-1.2.3.jar	1.2.3			
	xdoclet-exolab-module-1.2.3.jar	1.2.3			
	xdoclet-fr_FR-locale-1.2.3.jar	1.2.3			
	xdoclet-hibernate-module-1.2.3.jar	1.2.3			
	xdoclet-hp-module-1.2.3.jar	1.2.3			
	xdoclet-ibm-module-1.2.3.jar	1.2.3			
	xdoclet-java-module-1.2.3.jar	1.2.3			
	xdoclet-jboss-module-1.2.3.jar	1.2.3			
	xdoclet-jdo-module-1.2.3.jar	1.2.3			
	xdoclet-jmx-module-1.2.3.jar	1.2.3			
	xdoclet-jsf-module-1.2.3.jar	1.2.3			
	xdoclet-libelis-module-1.2.3.jar	1.2.3			
	xdoclet-macromedia-module-1.2.3.jar	1.2.3			
	xdoclet-mockobjects-module-1.2.3.jar	1.2.3			
	xdoclet-mvcsoft-module-1.2.3.jar	1.2.3			
	xdoclet-mx4j-module-1.2.3.jar	1.2.3			
	xdoclet-objectweb-module-1.2.3.jar	1.2.3			
	xdoclet-openejb-module-1.2.3.jar	1.2.3			
	xdoclet-oracle-module-1.2.3.jar	1.2.3			
	xdoclet-orion-module-1.2.3.jar	1.2.3			
	xdoclet-portlet-module-1.2.3.jar	1.2.3			
	xdoclet-pramati-module-1.2.3.jar	1.2.3			
	xdoclet-pt_BR-locale-1.2.3.jar	1.2.3			
	xdoclet-solarmetric-module-1.2.3.jar	1.2.3			
	xdoclet-spring-module-1.2.3.jar	1.2.3			
	xdoclet-sun-module-1.2.3.jar	1.2.3			
	xdoclet-sybase-module-1.2.3.jar	1.2.3			
	xdoclet-tjdo-module-1.2.3.jar	1.2.3			
	xdoclet-web-module-1.2.3.jar	1.2.3			
	xdoclet-webwork-module-1.2.3.jar	1.2.3			
	xdoclet-wsee-module-1.2.3.jar	1.2.3			
	xdoclet-xdoclet-module-1.2.3.jar	1.2.3			
	xjavadoc-1.5-snapshot050611.jar	1.5-snapshot050611			

Ant Build Scripts

Ant build scripts for tools should use the jars in lams_build. The following setup is suggested.

The j2ee.jar file presents a problem. This is not available under the GPL so it is not deployed, and shouldn't be included in our source code if we can avoid it. But it is needed for the ant build to succeed. So the ant build will define where the j2ee.jar can be found on the developer's PC. In the example given, the j2ee.jar referenced comes from the MyEclipse directory. If you are not using MyEclipse, replace this path with the j2ee.jar that IDE uses.

Add the following definitions:

```
sharedlib=../lams_build/lib
j2eelibs=C:/Program Files/MyEclipse/eclipse/plugins/com.genuitec.eclipse.j2eedt.core_3.8.4/data/libraryset/1.4
```

Define the mysql driver as:

```
database.driver.file=${sharedlib}/mysql/mysql-connector-java-5.0.8-bin.jar
```

Add the libraries to the project classpath

```
<path id="all-libs">
        <fileset dir="${lib}">
                <include name="**/*.jar"/>
        </fileset>
        <fileset dir="${sharedlib}">
                <include name="**/*.jar"/>
        </fileset>
        <fileset dir="${j2eelibs}">
                <include name="**/*.jar"/>
        </fileset>
</path>

<path id="project.classpath">
        <path refid="all-libs"/>
        <!-- Java CLASSPATH should be added as the last item -->
        <!-- This property is supposed to be set in eclipse  -->
        <pathelement location="${java.class.path}" />
</path>
```

The compile tasks can then refer to all the libs using:

```
<classpath>
  <pathelement location="${build.classes.java}"/>
    <path refid="project.classpath"/>
</classpath>
```

Using the Libraries (Runtime)

The deployed third party libraries have been added to the "classpath" of the ear, by defining them in the application.xml file. This allows the web applications to assume that the libraries are available and we can change the locations, names and/or versions without having to update all the web-applications. Of course, a change to the version will require a retest of the system.

This is probably a JBOSS only setup which is unfortunate, but the other three choices were not selected for the following reasons:

- Specify all the libraries on the Class-Path entry in the MANIFEST.MF for the ear. Various entries on the web indicate that this may work, but Fiona was unable to get it to work in JBOSS 3.2.6. This was the preferred method so if anyone can get this to work, we will switch to this method. Changing to this method does not require any changes to the other projects.
- Put all the libraries in the JBOSS lib directory. This is the method used in the past. It does not make it clear which libraries we have added to the system, and potentially conflicts with other applications running on the same JBOSS server. Changing to this method does not require any changes to the other projects.
- Specify the libraries on the Class-Path entry in the MANIFEST.MF for each web application. This would require updating the web applications whenever we make a change to the jar files, which is an unnecessary overhead. Quite apart from the fact that developers will go quietly mad trying to specify all the dependent jars and will end up specifying all the jars, just in case.
  The main LAMS jar files (lams.jar, lams_contentrepository.jar, etc) are not included in the application.xml so they must be explicitly listed on the class-path entry in a web application's MANIFEST.MF. Explicitly listing the jars allows a quick check to see what functionality that a tool is using. As there are only a few jars, this is not onerous and hopefully developers won't list all of them "just in case", as they would with third party jars.

If we decide to start changing the names of these jars, then we will need to put them in the ear classpath.

For example: Class-Path: ./lams.jar ./lams_tool_imscp.jar ./lams_contentrepository.jar

Adding A Libary

If you wish to add a library, please follow these steps:

- Email the technical lead on the project to let them know that you are going to add the library, its license status and why you think it is useful as a common library. This allows the tech lead to check if there is any reason why it shouldn't be added such as we considered this one before but decided not to make it common as different people want different versions, etc. Once the tech lead has said okay, then go ahead.
- Update this document. Include the source location of the library (i.e. the address of the website from which downloaded the jar) and a short note on the license details. This will help us to manage the libraries.
- Add the library to lams_build/lib/<directoryname>
- If the library is to be deployed at runtime:
    - Add the library to the application.xml file
    - Add the library to the assemble-ear task in the build.xml file (in lams_build).

Why do it this way?

The use of user libraries has its drawbacks (like the full paths in the import file). However a number of setups were examined for our project structure and none were without faults.

All the set up of libraries using the user library or direct project references only affect the Eclipse build path. All builds must done using ant and the ant scripts do not use the Eclipse build path anyway! So this setup is a compromise of creating minimal extra projects, minimal extra work in the ant scripts and yet having a reasonable Eclipse build path.

Other possibilities were:

Use Direct References
Set up the third party jar files as build path exports in lams_build, and use direct project references in the other projects. Say your tool uses the content repository. You could set up your Eclipse project to directly reference the lams_build source code, the lams_build jars and the lams_content_repository source code.

This was seriously considered as an alternative. However it would require leaving many projects open in Eclipse (due to the dependencies across projects) and that will start to use up resources on the developer's PC.

Use An EAR Project

Eclipse has a special EAR project that models the classpaths used in an EAR file. It is designed to support Web and EJB files, and has partial support for sharing standard jar files.

In practice, all it does is set up a lot of project. Once set up, the developer has to maintain the project dependencies.

The new module would not help building (as that will still be done in an Ant script). It will allow MyEclipse to directly deploy the ear, but it will deploy an Eclipse build path ear, not an ant build ear, so developers may miss problems that occur in the ant build version.

If we want to manage it using a lot of project dependencies, we might as well just do it ourselves (as described in direct references).

Use lams_common Project
Reuse lams_common for the jars. After all, isn't this the common project? Yes, but having a separate build project allows other build tasks to be isolated. It also will remove the temptation to write code in lams_common that requires other modules, without explicitly defining the dependencies.