Aspire Development FAQ

How do I move source code to a another package/group ID?

The trick here is changing the package name in multiple places. It is easiest to let Eclipse handle most of it (via Refactor->Rename).

1. In Eclipse, change the package name from the Package Explorer by clicking on it, then pressing Alt-Shift-R.
    a. This will also change the import line in your source code AND your directory tree to match.
    b. If you don't use Eclipse's Rename, you will have to do these steps yourself.
        i. Don’t forget ALL your source files, including src/test/java and src/main/resources.
2. Change the implementation attribute in ComponentFactory.xml.
3. Change the group ID in the pom.xml.
4. Change the Private-Package section of the POM.
    a. Do NOT change the aspire.framework line.
5. Run the command: mvn clean install
    a. You may seem some new errors about missing dependencies (that were in the package you moved away from); solve those by adding the old package as a dependency in the POM.
6. Note that your JAR may still include a com.searchtechnologies section AS WELL AS your new path because Aspire includes com. searchtechnologies.aspire.framework in every distribution.

   Now the project that uses the above JAR needs to be adjusted for its new location:

   ```
   <component name="customComponent" subType="default" factoryName="com.newgroupid:aspire-custom-
   component"/>
   ```

7. In the Aspire config file that references your component, change the factory name; for example:

8. In the POM for the enclosing project, find the dependency block for your component and change its groupId:

```
<dependency>
   <groupId>com.newgroupid</groupId>
   <artifactId>aspire-custom-component</artifactId>
   <version>0.4-SNAPSHOT</version>
   <scope>jar</scope>
 </dependency>
```

9. In the distribution.xml file for the enclosing project, make sure that there is a dependencySet for the new package:

```
<dependencySet>
   <outputFileNameMapping>${artifact.artifactId}-${artifact.baseVersion}${dashClassifier?}.${artifact.
extension}</outputFileNameMapping>
   <unpack>false</unpack>
   <useTransitiveDependencies>false</useTransitiveDependencies>
   <outputDirectory>bundles/aspire</outputDirectory>
   <includes>
     <include>com.newgroupid:*</include>
   </includes>
 </dependencySet>
```

JUNIT returns "Class Not Found" error

Things to try:

- Run a "**Project/Clean...**" command (within Eclipse) on your project.
- Right-click on your component project and do "**Configure Build Path...**", then go to the "**Order and Export**" task and make any dependent projects are moved to the top, before the Java System Libraries.

Error When Checking Out A Maven Project From Subversion

(within Eclipse)

The error is:

```
Can't rename C:\Documents and Settings\Admin\Desktop\workspaces\aspire01\maven.1260664537843
```

You will need to delete the stray "maven.1260664537843" folder in your workspace and try again. This might need to be repeated a few times before it works.

Check out the Subversion Console and the Maven Console for more details.

IE may try and turn our XML files into feeds

Try to avoid using metadata tags which start with "<rss" or "<feed", e.g., <feederLabel>.

Basically, IE looks for certain tags at the beginning of a random XML file to determine if it's an RSS feed. If it is, it may do something special with it.

When using an RSS 2.0, 0.91, or 0.92 feed, the HTTP Content-Type header should be “text/xml”:

```
Content-Type: text/xml
```

Because this is a generic content-type, IE7 will scan the first 512 bytes to look for the “<rss” string that indicates that it is a feed. If the string is found, the file is considered a feed, and the feed preview is shown to the user. For example, this is a RSS 2.0 feed for the IE blog, and IE7 determines that it is a feed because of the highlighted string.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" ... >
```

IE may then report that the XML file can not be parsed as an RSS feed (or some such error).

How do I solve class loading errors when running my third party code in OSGI?

When running under just the component test bench, everything worked fine, but when running under the application test bench, a third-party JAR is unable to locate its own properties file.

Sometimes, third-party JARs use wacky methods for loading classes. In the case of the ROME libraries, do the following under OSGi in order for the ROME JARr to locate its own properties files:

```
// Reset the classloader for this thread to prevent issues finding the ROME properties file
// and classes
if (!Thread.currentThread().getContextClassLoader().equals(SyndFeedInput.class.getClassLoader())) {
        System.out.println("Setting ClassLoader");
    Thread.currentThread().setContextClassLoader(SyndFeedInput.class.getClassLoader());
} else {
     System.out.println("!!! ClassLoader equal");
}
```

How to load a class from an external JAR file

You need to create a classloader containing the current classloader and the JAR file you wish to use. Try the following code:

```
private void setClassloader(ArrayList<StringBuffer> jarFiles) throws AspireException {

        ArrayList<URL> urls = new ArrayList<URL>();

        // Bail if there are no files.
        if (jarFiles == null || jarFiles.size() == 0)
               return;

           for (StringBuffer file:jarFiles) {
            // Ignore empty paths
            if (Utilities.isEmpty(file.toString())) {
                      warn("Ignoring blank jar file specification in component JNDI configuration");
                      continue;
             }

             // Get the path to the file
             String fullDriverPath = getFilePathFromAspireHome(file.toString().trim());
             debug("Adding %s to classpath", fullDriverPath);

             // Check it exists
             File jarFile = new File(fullDriverPath);

                if(!jarFile.exists()) {
                      // Jar file doesn't exist
                      throw new AspireException("aspire.feeders.JMSFeederImpl.jar-not-found",
                                  "Unable to locate the configured Jar file. The parameter (\"%s\") is
probably incorrect. Full path = \"%s\"",
                                  file, fullDriverPath);
             }

             // Construct the url for the file and add it to the list of urls
             try {
                      urls.add(jarFile.toURI().toURL());
                } catch (MalformedURLException e) {
                      throw new AspireException("aspire.feeders.JMSFeederImpl.cant-parse-jar-file-name", e,
                          "Unable to convert the Jar file to a URL. The parameter (\"%s\") is probably
incorrectly specified. Full path = \"%s\"",
                          file, fullDriverPath);
        }
}

// Create a class loader with the new jars
_queueClassLoader = URLClassLoader.newInstance(urls.toArray(new URL[urls.size()]), this.getClass().
getClassLoader());
```

You can then either load the classes directly from the loader:

```
// Load the class
_queueClassLoader.loadClass(className);
```

Or set the current thread's class loader to be the one created:

```
// Add the jars in to the classloader for this thread
if (_queueClassLoader != null) {
        debug("Adding jars to classloader");
        Thread.currentThread().setContextClassLoader(_queueClassLoader);
}
```

Why am I getting an infinite loop when reading my component's configuration?

I ran into a subtle infinite loop, which I'd like to share with you.

First, my XPath for extracting components from a component-manager configuration is this:

```
final static AXPath componentXPath = new AXPath("/config/components/component");
```

And my system XML was this:

```
<config>
  <components>
    <component name="feeder" subType="default" factoryName="Aspire.RSSFeeder">
                                 . . .
      </config>
    </component>

    <component name="pipeline" subType="pipeline" factoryName="aspire.Application">
      <config>
        <components>
          <component . . . />
                                 . . .
        </components>
      </config>
    </component>
  </components>
</config>
```

The infinite loop was this:

1. Start the system manager and get /config/components/component.
2. Start all of these components.
3. Now, the pipeline manager needs to start all of its sub-components. So what XPath does it use? Well, it's: /config/components /component

What happened is that the pipeline, when it got the list of components, because of the leading "/", it actually accessed the components from the top-level of the configuration file, not the components nested within the pipeline-manager's configuration section.

And so, it started all of the components, which meant that it started another copy of itself! (When it then went again to /config/components/component, got the top-level components again, and then started another copy if itself again, etc. etc.)

The Moral of the story:

When accessing your configuration elements with XPATH, do not use a leading "/". If you do, you will access the configuration of the system manager all the way at the top.

You will be passed an "element" as your configuration, which is a <config> element nested within the overall system configuration file. If you are using XPaths to extract information from the element, a leading "/" will go all the way to the top.

(Note that none of this affects how the digester works. You should still use a leading "/" for simple-digesting.)

To fix the problem, change your XPATH to simply: components/component

Note there is no need to prefix it with "config", since we are already "inside" the config node.

Don't Create Your Stage Inside of Another Maven Project

This will create unnecessary and destructive <parent> tags inside your stage's pom.xml file. If you discover these, just delete the entire <parent> tag.

It is best to use the archetype to create the stage directly inside your Eclipse project.

Why do my bundles sometimes get a timestamp when I use "mvn assembly:assembly"

Under certain circumstances, the bundles copied to the output directoy are named with a timestamp:

```
13/05/2010  20:00         4,106,811  aspire-groovy-0.2-20100510.030504-6.jar
13/05/2010  20:00            95,416  aspire-rdb-0.2-20100510.030658-5.jar
13/05/2010  20:00         1,244,640  aspire-rdbfeeder-0.2-20100510.030720-3.jar
13/05/2010  20:00           172,085  aspire-tools-0.2-20100510.031037-5.jar
```

You can solve this by adding an **outputFilenameMapping** to the distribution.xml file:

```
<dependencySet>
       <outputFileNameMapping>${artifact.artifactId}-${artifact.baseVersion}.${artifact.extension}<
/outputFileNameMapping>
       <unpack>false</unpack>
       <useTransitiveDependencies>false</useTransitiveDependencies>
       <outputDirectory>bundles/aspire</outputDirectory>
       <includes>
             <include>com.searchtechnologies:*</include>
       </includes>
 </dependencySet>
```

How do I abort / terminate a job?

Call job.terminate(). This will set a flag which causes the job to be terminated once the current stage completes. See Terminating Jobs for more details.

Strange class issues when running bundles

I got a stange class loadeder issue when loading a component

```
Exception in thread "Thread-11" java.lang.LinkageError: loader constraint violation: when resolving method
   "com.searchtechnologies.aspire.services.ServiceUtils.getComponentServiceTracker(Lorg/osgi/framework
   /BundleContext;Ljava/lang/String;)Lorg/osgi/util/tracker/ServiceTracker;" the class loader (instance of
org/apache
   /felix/framework/ModuleImpl$ModuleClassLoader) of the current class, com/searchtechnologies/aspire
/framework
   /ComponentImpl, and the class loader (instance of org/apache/felix/framework
/ModuleImpl$ModuleClassLoader) for
   resolved class, com/searchtechnologies/aspire/services/ServiceUtils, have different Class objects for the
type
   org/osgi/util/tracker/ServiceTracker used in the signature
       at com.searchtechnologies.aspire.framework.ComponentImpl.getComponentServiceTracker(ComponentImpl.
java:515)
       at com.searchtechnologies.aspire.framework.BranchInfo.setPipelineManagerName(BranchInfo.java:70)
       at com.searchtechnologies.aspire.jobErrorHandler.JobErrorHandlerImpl.resubmitJob(JobErrorHandlerImpl.
java:432)
       at com.searchtechnologies.aspire.jobErrorHandler.JobErrorHandlerImpl.resubmitJobs(JobErrorHandlerImpl.
java:396)
       at com.searchtechnologies.aspire.jobErrorHandler.JobErrorHandlerImpl.run(JobErrorHandlerImpl.java:364)
       at java.lang.Thread.run(Unknown Source)
```

It turned out to be a missing import package. Adding **org.osgi.util.tracker** to the **<Import-Package>** of the pom solved it.

I've also seen this error:

```
 Exception in thread "Thread-5" java.lang.LinkageError: loader constraint violation: when resolving field
"NODE" the class loader
   (instance of org/apache/felix/framework/ModuleImpl$ModuleClassLoader) of the referring class, javax/xml
/xpath/XPathConstants,
   and the class loader (instance of <bootloader>) for the field's resolved type, javax/xml/namespace/QName,
have different Class objects for that type
       at com.searchtechnologies.aspire.framework.AXPath.getElement(AXPath.java:112)
       at com.searchtechnologies.aspire.framework.AXPath.getElement(AXPath.java:99)
       at com.searchtechnologies.aspire.framework.ComponentFactoryImpl.persistComponent(ComponentFactoryImpl.
java:377)
       at com.searchtechnologies.aspire.framework.ComponentFactoryImpl.registerComponent
(ComponentFactoryImpl.java:333)
       at com.searchtechnologies.aspire.application.ComponentManagerImpl.registerComponents
(ComponentManagerImpl.java:153)
       at com.searchtechnologies.aspire.application.ComponentManagerImpl.initialize(ComponentManagerImpl.
java:65)
       at com.searchtechnologies.aspire.framework.ComponentFactoryImpl.registerComponent
(ComponentFactoryImpl.java:328)
       at com.searchtechnologies.aspire.application.AspireApplicationComponent.startManager
(AspireApplicationComponent.java:177)
       at com.searchtechnologies.aspire.application.AspireApplicationImpl.autoStart(AspireApplicationImpl.
java:82)
       at com.searchtechnologies.aspire.application.AspireActivator.run(AspireActivator.java:60)
       at java.lang.Thread.run(Unknown Source)
```

Again, this was a missing import. I cleaned up the pom and added **javax.xml.namespace** to the **<Import-Package>**

One error not caused by the pom:

```
ERROR: Unable to start system bundle. (java.lang.ClassCastException: com.sun.org.apache.xerces.internal.dom.
CommentImpl cannot be cast to org.w3c.dom.Element)
java.lang.ClassCastException: com.sun.org.apache.xerces.internal.dom.CommentImpl cannot be cast to org.w3c.
dom.Element
       at com.searchtechnologies.aspire.distributed.discovery.DiscoveryManager.initialize(DiscoveryManager.
java:106)
       at com.searchtechnologies.aspire.distributed.DistributedCommunicationsManager.initialize
(DistributedCommunicationsManager.java:137)
       at com.searchtechnologies.aspire.application.AspireApplicationImpl.doAdditionalInitialization
(AspireApplicationImpl.java:69)
```

This was because of a comment in an XML file being read by the following code:

```
 Element discoveryMethodElement = (Element)discoveryMethodsFromConfig.item(index);
```

The only solution at the moment is to remove the comment.

Calling Components from other Components

See Accessing Other Components for details on how to programatically access one component from another.

Suppose you have a situation where one Aspire component needs to call another component, for example where the RDBFeeder needs to call the RDB connection pool. You need to be careful when declaring the import and export packages in the components pom file and the component dependencies.

Basically, OSGi acts as a broker between components which need to access certain java packages, and components which supply these packages to other components:

blocked URL

In order for OSGi to know how to "wire things up", it needs to know (through the <Export-Package> and <import-Package> tags in the POM files, exactly what java packages (not, not classes, but entire packages) are available to other components (<Export-Package>) and which ones are need to be accessed from other components (<Import-Package>).

Also, we would strongly recommend that you separate out the methods of your classes, which need to be exported, and include these in a Java "Interface" – and then put this interface into a separate java package – to better control the interface between your packages. This will help organize your thoughts about exactly how packages should communicate to each other. See aspire-rdb and the RDBMSConnectionPool (inside the com.searchtechnologies.aspire.rdb package) for an example.

Your *caller* bundles should also include a dependency of your *called* bundle in the pom file.

Typically if you get something wrong you'll see an exception like:

```
 Java.lang.ClassCastException: com.searchtechnologies.aspire.components.DatabaseLogger cannot be cast to com.
searchtechnologies.cpatools.Logger
```

At the point at which you try to access the component:

```
 //import declared
 import com.searchtechnologies.cpatools.Logger;

 //this is executed in a method within the initialize() method:
 String componentName = getStringFromConfig(config, "componentNameDBLogger", null);

 //this is executed in the process() method and throws the class cast exception
 Logger logger = (Logger) getComponent(componentName);
```

If you still have issues, other things to try:

- If you're using release 0.3 or below, make sure the *called* bundle is listed earlier in the <bundles> tag than the *caller*:

```
<config name="assets">
  <bundles>
    <filePattern>bundles/aspire/aspire-called-bundle.jar</filePattern>
    <filePattern>bundles/aspire/aspire-caller-bundle.jar</filePattern>
  </bundles>
  <components>
  .
  .
  </components>
</config>
```

If you're using version 0.4-SNAPSHOT or later, this shouldn't be an issue as Aspire will automatically load the dependent bundle first.

- Open up the JAR files for both bundles with WINRAR and check the directories to ensure that the one which is supposed to have the com.searchtechnologies.cpatools.Logger file does have it (the called) and that the one which is not supposed to have it (the caller), in fact does not have it.
- Also open up the JAR files and check the MANIFEST file to see if the imports from the <Import-Package> and <Export-Package> elements are correctly copied into the MANIFEST file.

Pom File Dependencies

When creating components, you'll often need to reference other Aspire components or third-party JAR files. In order to do this, you'll end up with one or more dependencies in the pom file, similar to this:

```
<dependency>
  <groupId>com.searchtechnologies</groupId>
  <artifactId>aspire-simplefeeder</artifactId>
  <version>0.4-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>com.searchtechnologies</groupId>
  <artifactId>aspire-ccd</artifactId>
  <version>0.4-SNAPSHOT</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>net.java.dev.rome</groupId>
  <artifactId>rome</artifactId>
  <version>1.0.0</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>1.4</version>
</dependency>
```

You should have added these in order to make the code compile. If you didn't, and you added JAR files directly to the build path in Eclipse, then shame on you. You'll regret it later.

However, making it compile is not the same as getting it to run within Aspire.

The first thing to note is that only one of the dependencies in the fragment above has the **<scope>** tag. If you don't specify this, the scope defaults to *compile*. The scopes are described below:

- *compile* - Include the third-party JAR in my component (i.e., the bundle JAR file will contain the third-party JAR file).

  > In the example above, "aspire-simplefeeder" behaves like a "third-party JAR." It will actually be included in your component just like any other JAR from someone else.

- *provided* - The packages I need will be provided by this other aspire-X component, which I have specified as "provided" (i.e., the bundle JAR file will NOT contain the third-party JAR file).
- *test* - This is only needed for testing.

Aspire (from 0.4 onwards), when attempting to load a component, assumes any *provided* dependency (with exceptions such as **aspire-application**, **as pire-services**, and **org.osgi**) is a Full Aspire Component which must be pre-loaded before the current component can be loaded. The Aspire built-in dependency loader is only for complete Aspire components (they must be specified as <packaging>bundle</packaging> in their POM). It will not work for ordinary JAR files. This is by design. Ordinary JAR files must be included in your component (as a compile-time dependency), or they must be exported by another component.

In versions 0.3 and earlier, you need to ensure that a dependent bundle is loaded first by ensuring the file appears first in you <bundles> tag. Again, third-party JAR files must be included in your component (as a compile-time dependency).

Enabling JMX Console to investigate memory issues with Felix

If you experience memory issues or JVM core dumps you can enable JMX on Felix and then connect using JConsole.

Just add the following parameters to the JAVA_OPTS in the start up script:

```
  -XX:+HeapDumpOnOutOfMemoryError -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=9999 -
Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
```

Restart Aspire and then use JConsole to connect to port 9999.

Aspire and permgen memory issues

The permgen is where the JVM keeps the actual classes (not the instances, just the "boiler plates"), including fields, method names, etc.

Given how OSGI works, exposing some classes as "private" within the bundle and other as "public", i.e., available to other bundles, then the classloader(s) could have different dynamic classes for what could be considered copies of the "same" class (which may or may not be true in reality, because they could be different versions of it, with even different functionality).

Hence, OSGI will tend to fill more easily this memory.

The more evident example today is the JMS Feeder; which uses reflection on its implementation, thus indirectly using more of this memory.

All in all, it is a small price to pay (increasing this memory) for all the capabilities we get from the OSGI framework.

Therefore, it is important that we monitor this memory usage. It can be checked using the JConsole.

For setting and/or changing the permgen size, just add/change the MaxPermSize **(valid for JAva 1.7 and below)** parameter to the JAVA_OPTS in the start up script. An example where we set the size to 256m is below:

```
-XX:MaxPermSize=256m
```

If you are using Java 1.8 try setting and/or changing the metaspace size, just add/change the -XX:MaxMetaspaceSize parameter to the JAVA_OPTS in the start up script. An example where we set the size to 256m is below:

```
-XX:MaxMetaspaceSize=256m
```

⚠ Given its nature, permgen nor metaspace is garbage collected.

Pipeline manager thread pool and queue are full, but all processing is stuck

In some rare cases, an error in the configuration of pipeline managers and branches can cause a deadlock. To prevent that, be sure that you are not branching sub-jobs to the same pipeline manager doing the branching.

Incorrect

Pipeline Manager A: XMLSubjobExtractor -> PrintToFile

Where XMLSubjobExtractor branch is configured to send jobs to Pipeline Manager A. This will almost certainly cause a deadlock.

Correct

Pipeline Manager A (parent jobs): XMLSubjobExtractor

Pipeline Manager B (sub-jobs): PrintToFile

Where XMLSubjobExtractor branch is configured so it sends jobs to Pipeline Manager B.

Notice that the same may happen wherever you have a branch configuration. So be careful when setting the destination of your branches.

Parsing Errors

```
org.xml.sax.SAXParseException: Content is not allowed in prolog.
```

This error is caused by the presence of a byte-order mark (BOM) at the start of your XML file. You can see the BOM using "od -c <filename>". You can remove the BOM simply by copying and pasting the entire file contents to a new file in a text editor since the BOM does not get copied to the clipboard.