Accessing Other Components

Sometimes you may need a component that accesses other components.

Examples

- Getting an RDB connection from the "RDB Connection" component
- Accessing a Lucene index from the aspire-lucene component
- Reading the Content Control Database from the CCD component
- Reading from a hash table

The standard method within OSGi for accessing other components in the system is to use an OSGi service tracker. This will ensure that your stage always gets the latest version of the component, even if the component is released and refreshed. Since in Aspire, it will be a component that wants to talk to another component, the component maintains a cache of OSGI service trackers to other components and exposes this by way of a method call.

When your component wants to access another, you call that method, passing the path to the component you want. If there's a service tracker for the requested component, it's used to get a reference to the component. If there isn't a service tracker is created and added to the cache. The new tracker is then used to get the reference.

However, there are a number of steps you'll go through to get access to another component.

On this page:

Step 1: Add a config element for the name of the stage you need to access

Inside your component, you'll need to have a configuration element which specifies the name of the stage to access. It might look something like this:

The following is a sample configuration for **your** component - i.e. the one which is calling the shared resource.

```
<component name="testComponent" subType="default" factoryName="aspire-mycomponent">
    <componentRef>/systemGetCcdTest/CCDService</componentRef>
    .
    .
    .
</component>
```

Example Configuration

testComponent want to access */systemGetCcdTest/CCDService*.

This name then needs to be loaded into your stage in your initialize() method, and stored for when you want to use it. You can use the getStringFromConfig() method to access the component name, as follows:

```
componentRef = getStringFromConfig(config, "componentRef", null);
```


In the example, "componentRef" will need to be a member variable on your stage.

The Other Component's Interface

Typically, the reason why you need to access another component is to use some additional methods that the component supplies. These additional methods will be presented in an interface that the component exports for your use.

Component	Supported Interface(s)
RDB Connection	RDBMSConnectionPool
Embedded Lucene Search Engine	AspireLucene

Hash Table	AspireHashTable

What this means that once you get a reference to the component, you can cast the component's object to the specified interface, and then start using the exported methods. For an example of how this is accomplished, see below.

The above list is only a small, partial list of interfaces supported by Aspire components. See the javadoc for a complete list of all interfaces supported by all official components.

Step 2: Add the other component as a dependency to your pom.xml file

You will need to add the other component as a "provided" dependency in your pom file. For example:

```
<dependency>
  <groupId>com.searchtechnologies</groupId>
  <artifactId>aspire-lucene</artifactId>
  <version>0.4.1</version>
  <scope>provided</scope>
</dependency>
```

This dependency will be used in two ways:

1. When you run your unit tests, JUnit and Eclipse will make the other component available as a Jar file so you can use it's interfaces and methods as necessary for your unit testing.
2. When Aspire loads your component it does dependency checking by looking in your component's pom.xml file. All "provided" dependencies to other components that you have will be automatically loaded *before* Aspire loads your component.

Step 3: Import the other component's Java packages

OSGi will also need you to identify what packages from the other component you will be using. OSGi uses this information to correctly "wire up" your component so that when you need classes from these packages, it will know how to get them.

This is done by adding java packages to the <Import-Package> tag inside your pom.xml file, as shown below:

```
 <Import-Package>
  com.searchtechnologies.aspire.services,
  com.searchtechnologies.aspire.lucene,
  javax.xml.*,
  org.osgi.framework,
  org.osgi.service.log,
  org.osgi.util.tracker,
  org.w3c.dom,
  org.xml.*,
  org.apache.lucene.*
</Import-Package>
```

The first import above is the interface implemented by the component which produces AspireLucene objects. This is the most usual situation, using other components for very specific interfaces which they provide.

The second import above allows access to all of the org.apache.lucene. Some components (such as aspire-lucene) do provide access to third party libraries. If you need to use these third party libraries, you should use the techniques above to pull in the appropriate component (such as aspire-lucene) as a dependency.

Step 4: Access the other component

When you access the other component, use the "getComponent(name)" method of the component implementation. This method uses the cache of service trackers and is extremely fast, since basically it just returns the reference to the component which the service tracker has stored internally.

Therefore, the following code should be included in the process() method for your stage, and *not* the initialization method. The results of the getComponent(name) method should only be stored in local variables, and should not be persisted in any way

However, it's possible that the other component will be (temporarily) unavailable. This can occur if the other component has not yet been initialized, or if it is in the process of being refreshed. The getComponent(name) method handles this senario, and waits for the component to become available. If you want to control the amount of potential wait time, use the getComponent(name, timeout) method

The example below shows how one would get an RDBMS connection pool from the RDB component. If the component is not available, we wait until it is. The default timeout is "AspireApplication.OSGI_TIMEOUT".

```
RDBMSConnectionPool rdbConnectionPool = (RDBMSConnectionPool) getComponent(componentRef);
```

This call will either return a reference to the component, or throw an exception if it's not available.

componentRef is a String containing the path to the component you wish to access - /myAppliation/myRDB