



adaptTo()

APACHE SLING & FRIENDS TECH MEETUP
2 - 4 SEPTEMBER 2019

OSGi best practices
Christian Schneider Adobe

- Computer scientist at Adobe
- Apache member and committer
- Twitter [@schneider_chris](https://twitter.com/schneider_chris)
- Website liquid-reality.de



Agenda

- Create bundles the lean way
- How to avoid Start Levels
- Best practices around DS
- Loose coupling and easy application assembly

Creating bundles

How you used to create bundles

- Maven bundle plugin
- Detailed setup in each pom.xml
- Explicit export package with version



Error prone

Breaks with refactoring

How you used to create bundles

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <instructions>
      ...
    </instructions>
  </configuration>
</plugin>
```



How you used to create bundles

```
<Export-Package>
  org.apache.karaf.diagnostic.management
</Export-Package>
<Import-Package>
  com.sun.management*;resolution:=optional,
  *
</Import-Package>
<Private-Package>
  org.apache.karaf.diagnostic.command,
  org.apache.karaf.diagnostic.common,
  ...
</Private-Package>
<Bundle-Activator>
  org.apache.karaf.diagnostic.internal.Activator
</Bundle-Activator>
```



[Source: Apache Karaf diagnostic plugin](#)

Create bundles the lean way

- Use bnd-maven-plugin only in parent
- In each bundle define exports, requirements and capabilities using OSGi R7 annotations
- Imports are handled automatically
- Use bnd.bnd file only in case you need manual override

Exporting a package

package-info.java

```
@org.osgi.annotation.versioning.Version("1.2.0")
```

```
@org.osgi.annotation.bundle.Export
```

```
package my.package;
```

Use Semantic versioning

When do you need to increase a package version?

Hard to do manually

- Configure semantic versioning plugin
- Current API is compared to last release
- Build failure when a package export version increase is needed

Start order and services

Why is start order frowned upon in OSGi?

- Allowing any order give the system room for optimization
- Bundles can be uninstalled and installed at any time
- Deadlocks on forced order

React on services instead of enforcing start order

Why people use start order

- Code requires certain state of system
- How to make sure this state is reached?

Start order looks like a good solution (at first)

How to not use services

```
ref = context.getServiceReference(HealthCheck.class);  
service = context.getService(ref);
```



Why is this bad?

- Have to check for null
- Service may be not (yet) registered
- If you repeat and wait you block threads
- You need to unget the service after use

Is ServiceTracker better ?

```
tracker = new ServiceTracker<HealthCheck,  
HealthCheck>(context, HealthCheck.class, null);  
tracker.getService();
```



Not better than context.getService() !

We need something reactive !

Component best practices

Always use a DI framework in OSGi

```
@Component
```

```
Class MyClient {
```

```
    @Reference MyService myService;
```

```
}
```

Let Declarative Services solve this for you.

Internal components and wiring

- DS can only bind to services.
- How do I keep a service internal to a component?

Use a service class or interface from a private package

Never block in @Activate

@Activate

```
public void activate(BundleContext context) {  
    executor.execute() -> {  
        // Do some long running stuff  
        context.registerService(Servlet.class, this, new Hashtable<>());  
    };  
}
```

Make blocking calls async and export service by hand.

Be careful with adaptTo()



What is adaptTo()

- By coincidence a name of some conference 😊
- Popular pattern in sling

```
Session session = resolver.adaptTo(Session.class);
```

Great pattern for simple cases but can have issues



Be careful with adaptTo()

- AdapterFactory services might not (yet) be present
- Some adaptations get OSGi services via registry
- Timing issues like to context.getService()

Replace adaptTo() by service references if possible



Be careful with adaptTo()



```
PageManager pageManager = resolver.adaptTo(PageManager.class);
```

null?
What now?



Service reference instead of adaptTo()

@Reference

```
private PageManagerFactory pageManagerFactory;
```

```
...
```

```
PageManager pageManager =
```

```
pageManagerFactory.getPageManager(resolver);
```

Always safe to use

Loose coupling and easy application assembly

Loose coupling vs application assembly

Loose Coupling

- Depend on interfaces not implementation
- Whiteboard pattern achieves even better decoupling

Application is loosely coupled
but
not enough information to find the bundles
with service impls or whiteboards

Assembly

- Determine list of bundles to install
- Should work with minimal definitions and resolver

```
@Component(  
    service = Servlet.class,  
    property = "osgi.http.whiteboard.context.path=/ myservlet "  
)  
public class MyServlet extends HttpServlet {  
    ...  
}
```

- Achieves loose coupling
- Assembly of application difficult

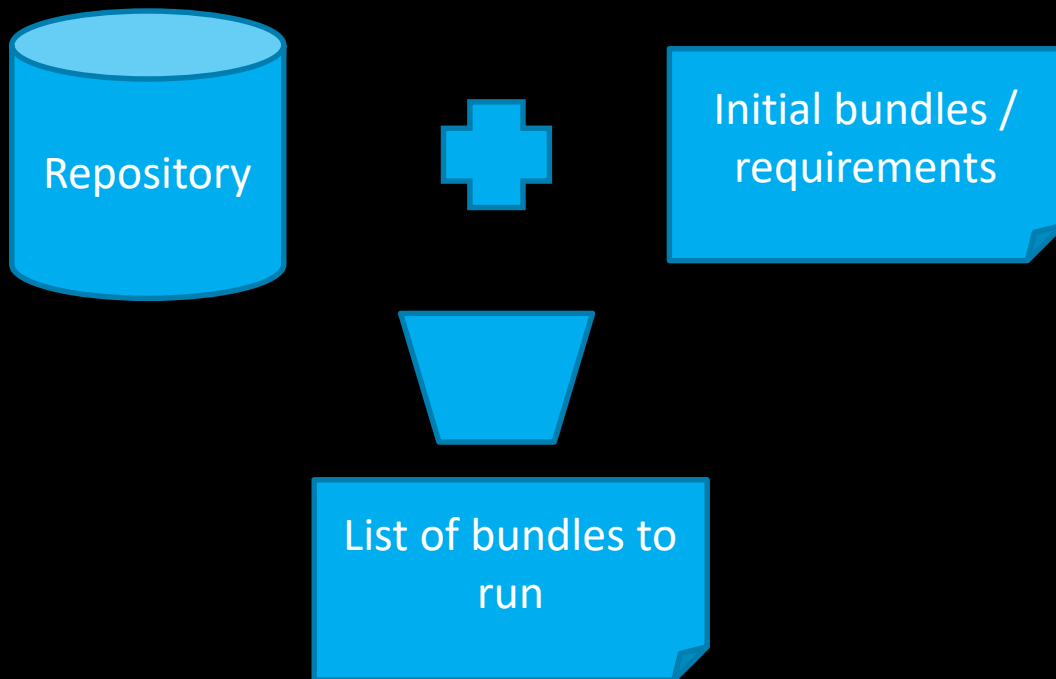


New Http Whiteboard

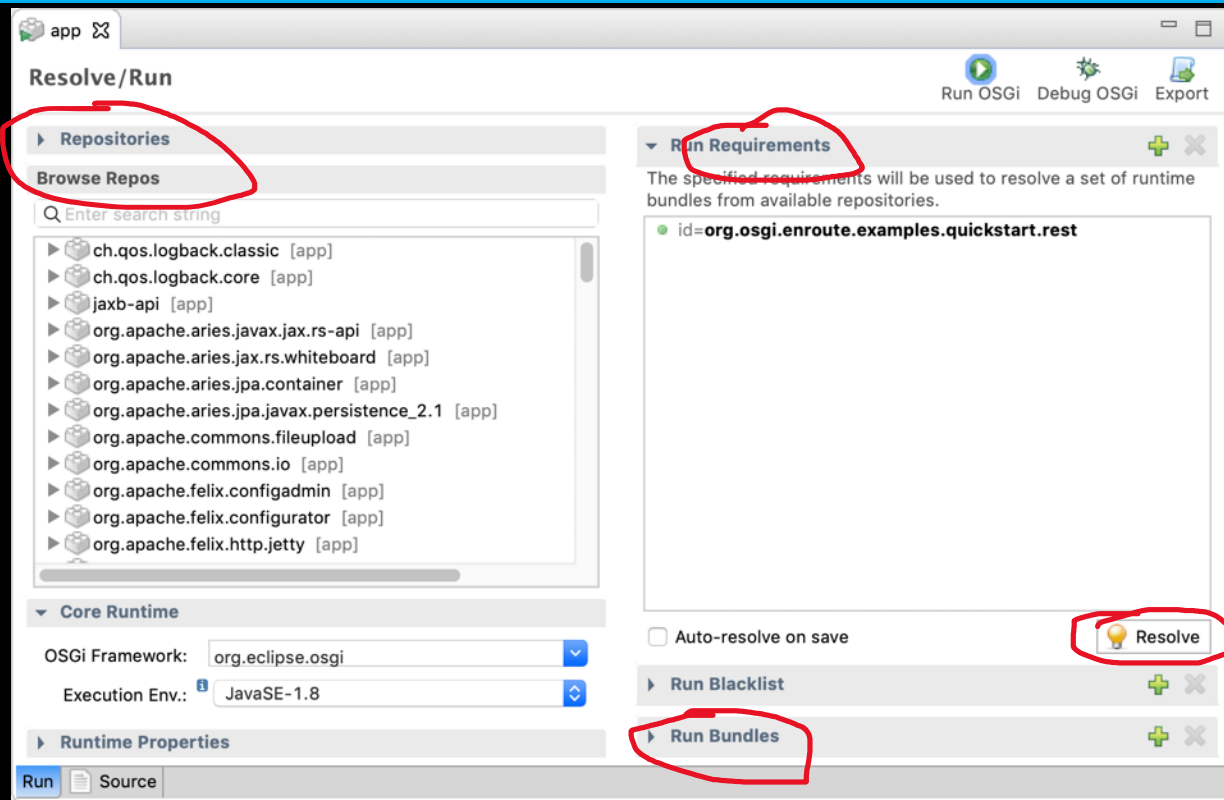
```
@Component(service = Servlet.class)
@HttpWhiteboardServletPattern("/myservlet")
public class MyServlet extends HttpServlet {
    ...
}
```

New annotations: Easier configuration + requirements

Requirements & Capabilities drive assembly



Assembly in practice



The screenshot shows the Eclipse IDE's 'Resolve/Run' dialog for an OSGi application. The dialog is titled 'app' and has tabs for 'Run', 'Source', and 'Properties'. The 'Run' tab is active. The 'Resolve/Run' section is expanded, showing a list of repositories and a list of core runtime bundles. The 'Run Requirements' section is also expanded, showing a single requirement: 'id=org.osgi.enroute.examples.quickstart.rest'. The 'Resolve' button is highlighted with a red circle. The 'Run Bundles' section is also highlighted with a red circle.

Repositories

- ch.qos.logback.classic [app]
- ch.qos.logback.core [app]
- jaxb-api [app]
- org.apache.aries.javax.jax.rs-api [app]
- org.apache.aries.jax.rs.whiteboard [app]
- org.apache.aries.jpa.container [app]
- org.apache.aries.jpa.javax.persistence_2.1 [app]
- org.apache.commons.fileupload [app]
- org.apache.commons.io [app]
- org.apache.felix.configadmin [app]
- org.apache.felix.configurator [app]
- org.apache.felix.http.jetty [app]

Core Runtime

OSGi Framework: org.eclipse.osgi

Execution Env.: JavaSE-1.8

Run Requirements

The specified requirements will be used to resolve a set of runtime bundles from available repositories.

- id=org.osgi.enroute.examples.quickstart.rest

☐ Auto-resolve on save

Resolve

Run Bundles

Questions?

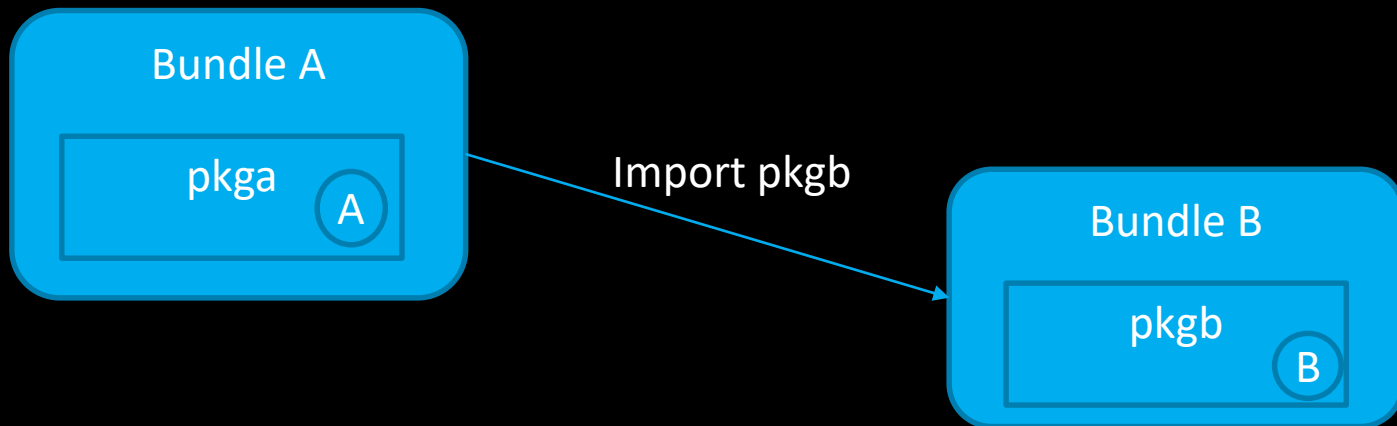
- Follow me on Twitter [@schneider_chris](https://twitter.com/schneider_chris)
- Demo project
 - <https://github.com/cschneider/osgi-best-practices>

More links in the Demo project

Backup

Everything you need to know about OSGi class loading

OSGi class loading



What happens when you do new B() in OSGi?

@Component

Class A {

@Activate

void loaderTest() {

ClassLoader clA = this.getClass().getClassLoader();

B b = new B();

ClassLoader clB = b.getClass().getClassLoader();

}

}

What is the
ClassLoader of
A and B and why?

OSGi class loading

- Manifest defines imported package and version ranges
- Resolve wires each imported package to a bundle providing the package
- On loading a class these locations are checked
 1. Bound imports
 2. Contents of the bundle
 3. bootstrap ClassLoader

Classloading is **delegated** to other bundle



So class B is loaded by ClassLoader of
BundleB

A Class is always loaded by the
ClassLoader of its own bundle