Tutorial

Florian Waibel, Johannes Eickhold, Markus Knauer
Survey

Who has used...

- PDE or Bndtools to build bundles
- Ant / Maven Tycho / Gradle to automate building of bundles
- ServiceTracker / DS / DS with annotations
Why “Kitchen Talk”? 
Who we are

Florian      Johannes      Markus
Roadmap

1. Install IDE + Bndtools
2. Tooling Walkthrough
   ○ Tasks 1 - 3
3. “Hands On” the hot “new” OSGi stuff
   ○ Task 4 - 5: Declarative Services (DS)
   ○ Task 6: ConfigurationAdmin Service
   ○ Task 7: HTTPService
Installing Prerequisites

1. Eclipse Luna SR2 RCP/RAP Package
2. Bndtools

---

pre-packaged versions are available!
Download Eclipse

Goto http://download.eclipse.org/~mknauer/osgi/

and download prepackaged Eclipse archive depending on OS

USB stick:

    cp /eclipse-rcp-luna-SR2-bnd-win32-x86_64.zip ~/
Install Eclipse

- Eclipse Luna (4.4.2) SR2 Packages
- BNDtools 4.2.1

http://bndtools.org/installation.html

USB stick:
unzip eclipse-rcp-luna-SR2-bnd-win32-x86_64.zip
tar zxf eclipse-rcp-luna-SR2-bnd-linux-gtk-x86_64.tar.gz
Get Git Repo

```
git clone https://github.com/eclipsesource/osgi-tutorial.git
```

USB Stick (Get local copy of the Git repository)

```
unzip /usb/osgi.tutorial_GITrepo.zip -d ~/git/
```
Tutorial as Branches

During the Tutorial YOU do:

1. Try to solve the tasks (Hint: Look for TODO task_x.y in the code)
2. `git diff task_x_<task_name>_final`
3. `git checkout task_x+1_<task_name>_start`
Kickstart - Import Projects

1. Start Eclipse on **new and empty** workspace
2. Open Git Perspective
3. “Add an existing Git Repository to local WS”
   - Select `~/git/osgi-tutorial`
4. Context Menu → Import Projects…
5. Open Bndtools perspective
Ready, Steady, ... Go!
1 Bndtools + OSGi + Console

- First contact with Bndtools
- Use the mighty Gogo shell
- Discover services in minimal Equinox
Bndtools

Minimal tooling to create OSGi bundles

Global settings in cnf project

Bundle repositories (like target platforms)

.bnd files define bundles (one or more!)

.bndrun files ⇔ Eclipse launch config
Bndtools - Repositories

Bundles from projects

Local prepopulated with Eclipse Luna SR2

Can be remote: “Bndtools Hub” is hosted on GitHub
Equinox OSGi console

You want this:

osgi>

Use Felix Gogo bundles:
org.apache.felix.gogo.command
org.apache.felix.gogo.runtime
org.apache.felix.gogo.shell
GoGo Shell Commands

lb  list bundles, use -s to see symbolic names
inspect capability service <bundle id>  show all services provided by a bundle
start/stop  start and stop bundles
grep  same as Unix command (use with pipe | )
Start 1 - Use the Force (tooling)

git checkout task_01_helloworld_start
Import... -> new project
1.2 Launch Equinox from `equinox.bndrun`.

To run, resolve dependencies of these bundles, and ensure they will run.
1.3 Enable Console

- add this to `equinox.bndrun` (src tab)

```bash
-runproperties: \
  osgi.console=,\n  osgi.console.enable.builtin=false,\n  activate.lazy.bundles=org.eclipse.equinox.console
```

- or use editor (run tab)
1.4 Use Commands

1. List all running bundles
2. Show their symbolic names  
   (Hint: use help <command>)
3. List services in System Bundle (org.eclipse.osgi)
4. Use grep to find services related to logging  
   (Hint: Pipe ‘|’ is your friend)
git diff task_01_hello_world_final
2 Oldschool OSGi with Bndtools

1. Implement your first bundle:
   ○ A simple bundle with a BundleActivator saying “Hello OSGi”, “Goodbye OSGi”

2. Implement your second bundle:
   ○ Listen to LogService events of other bundles starting / stopping
Bundle Descriptor: .bnd file
The LogService

- Equinox provides basic infrastructure for logging in bundle: `org.eclipse.osgi`
  - No additional bundles required!
- `org.osgi.service.log.LogService` [javaDoc](https://www.eclipse.org/osgi/)
  - Provides interface for bundles to write messages
  - Defines log levels
- `org.osgi.service.log.LogReaderService` [javaDoc](https://www.eclipse.org/osgi/)
  - Get log entries
  - Add / Remove LogListeners
2.1 Implement your First Bundle

1. Implement simple dummy: `c.e.o.t.t.a.Activator`

2. One class, only implement `BundleActivator`

3. In `start()` say "Hello OSGi"

4. In `stop()` say "Goodbye OSGi"
Start 2 - Create Bundles

git checkout task_02_bundles_start
Import → new project

(Hint: You might stash/commit local changes)
2.2 Second Bundle: LogChecker

1. Already implemented: `c.e.o.t.t.b.LogChecker`
   Make it a functional bundle!

2. Set “Private Packages” and “Activator”
   in file `b-logChecker.bnd`

3. Refresh bundle and watch console
2.3 Third Bundle: a LogListener

1. Partly implemented: `c.e.o.t.t.c. ConsoleLogListener`
   Complete it: attach LogListener to all LogReaders

2. Make sure this information is logged:
   ○ Log-level
   ○ Timestamp
   ○ Bundle’s symbolic name
   ○ Log-message
2.4 Verify: three bundles working

1. Bundle \textit{a} says hello and good bye
   ○ Start and stop it!

2. Bundle \textit{b} says: \textit{“LogService is available”}

3. Bundle \textit{c} logs all required information
   ○ level, timestamp, symbolic name, message
End 2 - created some OSGi bundles

```
git diff task_02_bundles_final
```
3 Unit and Integration Testing

- Test your code
- Test your bundle wiring/interaction
  - executed in OSGi stack
- Integration into CI
Bndtools Support out-of-the-Box

- Unit tests with JUnit 3 & 4
  - Test folder in regular project
  - Unit tests are not part of a bundle!
- Integration tests
  - Separate project
  - Special project template
  - Use “Run As” → “Bnd OSGi Test Launcher (JUnit)"
- Integrated into default gradle build
  ⇒ offline build / CI
Build Dependencies

Managed in bnd.bnd file → “Build tab”

To satisfy projects with tests, add...

- `${junit}` and `org.mockito.mockito-all`
  to build dependencies
- When a bnd project is created by wizzard, the build dependencies are already in place!
Create Integration Test Project

Ctrl + n

right click
Execute Integration Test Project
Execute Integrationtest Project
Start 3 - Write and Execute Tests

```
git checkout task_03_testing_start
Import → new project
```

Attention: Project contains error markers!
3.1 Fix Build Dependencies

1. On the **Source tab** of `bnd.bnd`:
   - provide the missing build dependency
     - Add `$\{junit\}$

2. **Switch to the Build tab**
   - Use the green + to add the “mockito-all” bundle
3.2 Implement Unit Test

In ConsoleLogListenerTest.java

1. Implement the TODO
   ○ Uses Mockito
   ○ If you are unsure, peak into the solution:
     solution is on branch task_03_testing_start

2. Execute unit tests in IDE (Run As → JUnit Test)
3.3 Implement Integration Test

In c.e.o.t.i.ExampleTest.java

1. Look at the two implemented tests
2. Implement the third test
3.4 Verify: All Tests are Green

1. Fix build dependencies
2. Fix and run unit test
3. Fix and run integration test
4. Execute: ./gradlew clean check
End 3 - Unit and Integration Tests

git diff task_03_testing_final
4 DS with Annotations

Java → `<XML />`

at build-time

`@Component` component `component.xml`
@Component
public class Foo {
  @Activate
  public void init() { /* do some initialization */ }
  @Deactivate
  public void shutdown() { /* do some cleanup */ }
  @Reference
  public void setBar(Bar bar) { this.bar = bar; }
  ...
}

<scr:component xmlns:scr="..." name="foo" activate="init" deactivate="shutdown">  
  <implementation class="c.e.o.t.Foo"/>  
  <reference name="bar" interface="c.e.o.t.Bar" bind="setBar" />  
</scr:component>
Bndtools’ Bundleexplorer
How to Open the Bundleexplorer?

Double click the blue bundle version or use bundle in
Apache Felix Web Console

Bundle: org.apache.felix.webconsole

http://localhost:8080/system/console/

User/PW: admin/admin
Introduction XRay

Bundle: aQute.xray.plugin

http://localhost:8080/system/console/xray
Start 4 - Declarative Services

```
git checkout task_04_ds_start
Import → new projects
```
4.1 Complete Implementation

Look for TODOs in the service implementations:

WaitressImpl, CookImpl and KitchenWhiteboardImpl

Hint: Get unit tests green
4.2 Feed the hungry customer

- Wire the components `WaitressImpl`, `CookImpl` and `KitchenWhiteboardImpl`
- Only use `@Component` and `@Reference`
- Implement all `TODOs` to see the kitchen in action

Hint: Get the integration tests green
4.3 Verify: DS works via Annotations

1. Services are implemented
   ○ *unit tests* are *green*

2. Services are wired correctly via annotations
   ○ *integrations tests* are *green*

3. Look at the services in the WebConsole!
End 4 - Used DS Annotations

```bash
git diff task_04_ds_final
```
5 Create custom Gogo commands

Gogo shell picks up Services with specific properties

```java
@Component(
    property = {
        CommandProcessor.COMMAND_SCOPE + " :String=tutorial",
        CommandProcessor.COMMAND_FUNCTION + " :String=order",
        CommandProcessor.COMMAND_FUNCTION + " :String=deliver"
    },
    service = Object.class
)
```
Start 5 - Gogo Shell Commands

git checkout task_05_commands_start
5.1 Talk to your App on the Shell

1. **Use** `WaitressCommands`, `KitchenWhiteboardCommands`, **and** `KitchenCommands` **to expose the API introduced in task 4 as Gogo shell commands**

2. **Wire** `Waitress` **via DS with annotations in the WaitressCommand (same for Kitchen)**
Verify: Talk to your App on the Shell

osgi> order
osgi> order
osgi> cook
osgi> deliver
osgi> kitchen:info
1 meal(s) ordered / 0 meal(s) ready

(Hint: kitchen:info command is already implemented)
End 5 - Added new Shell Commands

git diff task_05_commands_final
6 Serving Multiple Meals

Config Admin

...and allow to update the meals at runtime.
Extended Interfaces...

```java
public interface Waitress {
    void order(String meal);
    void deliver(String meal);
    ...
}

public interface Kitchen {
    void cook(String meal);
    ...
}

public interface KitchenWhiteboard {
    void incrementReadyMeals(String meal);
    void decrementReadyMeals(String meal);
    void incrementOrders(String meal);
    void decrementOrders(String meal);
    int getReadyMealsCount(String meal);
    int getOrdersCount(String meal);
    List<String> getAvailableMeals();
    ...
}
```
...and Implementation Classes

```java
public class KitchenWhiteboardImpl implements KitchenWhiteboard, ManagedService {
    private final List<String> meals;
    private final Map<String, AtomicInteger> mealReadyCounters;
    private final Map<String, AtomicInteger> orderCounters;

    public void incrementReadyMeals(String meal) {
        mealReadyCounters.get(meal).incrementAndGet();
    }

    public List<String> getAvailableMeals() {
        return Collections.unmodifiableList(meals);
    }
}
```
Managed Service

```java
public class KitchenWhiteboardImpl
    implements KitchenWhiteboard, ManagedService {
...
    public void updated(Dictionary properties)
        throws ConfigurationException {
        if (properties != null) {
            ...
        }
    }
}
```
@Component(
    configurationPolicy = ConfigurationPolicy.REQUIRE,
    configurationPid = KitchenWhiteboardImpl.COMPONENT_PID
)

public class Foo implements ManagedService {
    @Override
    public void updated(Dictionary properties) throws ConfigurationException {
        ...
    }
    ...
}

<scr:component name="foo" configuration-policy="require" deactivate="deactivate"
configuration-pid="com.eclipsesource.osgi.tutorial.KitchenWhiteboardImpl">
    <implementation class="c.e.o.t.Foo"/>
    <service>
        <provide interface="org.osgi.service.cm.ManagedService"/>
    </service>
</scr:component>
How to Handle Configurations

How to retrieve a Configuration?

```java
Configuration configuration = configurationAdmin.getConfiguration("some PID");
```

How to alter a Configuration?

```java
Hashtable<String, Object> properties;
String[] values = {"foo", "bar");
properties.put(Property Name, values);
```

How to update a Configuration?

```java
configuration.update(properties);
```
Start 6 - Configuration Admin

```bash
git checkout task_06_config_start

Import → new projects
```
6.1 Make Whiteboard ManagedService

In file `KitchenWhiteboardImpl`

1. Add configuration policy and PID
2. Add null check for optional Dictionary parameter
3. Use MEALSPROPERTY from Dictionary to update the meals list
4. Make sure counters for all meals are available in Maps
6.2 Provide Initial Configuration

In file KitchenWhiteboardConfigurator

1. Wire ConfigurationAdmin via DS with annotations
2. Get configuration of KitchenWhiteboardImpl's PID
3. "Burger" and "Salad" as initial meals to the configuration
4. Call update on the configuration
6.3 Add Meals via WebConsole

http://localhost:8080/system/console/configMgr
End 6 - Integrated Config Admin

git diff task_06_config_final
Servlets can be registered at the HttpService via the whiteboard pattern

Add / publish servlets by just annotating them

Learn how to wire our DS services with servlets
import org.osgi.service.http.HttpService;
...

@Activate
public void init() throws Exception {
    httpService.registerServlet("/dashboard/*", new KitchenWhiteboardServlet(kitchenWhiteboard),
                            null, null);
}
Start 7 - HttpService + Whiteboard

git checkout task_07_http_start
7.1 Replace the **active** Registration

```java
@Component(
    property = {
        HttpWhiteboardConstants.HTTP_WHITEBOARD_SERVLET_PATTERN + "+ =/<context>/*" },
    service = Servlet.class
)
public class MyServlet extends HttpServlet {
...
}
```
7.2 Register Filter via Whiteboard

@Component(
    property = {
        HttpWhiteboardConstants.HTTP_WHITEBOARD_FILTER_PATTERN
        + "=/<context>/*" }
)

public class MyFilter implements Filter {
...
}

Log all access to the Servlet
7.3 check results....

osgi> Request from remote address 0:0:0:0:0:0:0:1.
End 7 - Servlets via HttpService

git diff task_07_http_final
Congratulations, you made it!
Thank you!
Evaluate the sessions

Sign in: www.eclipsecon.org
Additional Information
OSGi Declarative Services (DS)

Around since 4.0 + YEAR?
uses XML for the wiring at run-time
Tooling? Eclipse PDE Component Editor?

Annotation-based since SPEC VERSION + YEAR?
DS Annotations generate the XML at build-time where the other DI frameworks interpret them at run-time
@ConfigurationPolicy - Configuration Policy for the Component annotation.

@Modified - Identify the annotated method as the modified method of a Service Component.

@ReferenceCardinality - Cardinality for the Reference annotation.

@ReferencePolicy - Policy for the Reference annotation.

@ReferencePolicyOption - Policy option for the Reference annotation

(Hint: not needed for the next item)
Comparison: DS Annotations

**Felix SCR Annotations**
- Inheritance support for abstract components
- Generated bind/unbind methods

**BND/OSGi DS Annotations**
- Boilerplate for the new OSGi DS Annotations in the 4.3 Release
- Fully supported by the Maven Bundle Plugin
- No other library dependencies
Managed Services

Managed Services are OSGi services that need access to configuration data at initialization and/or run-time. These configurations are administered through the OSGi Configuration Admin service.
"The Configuration Admin service is an important aspect of the deployment of an OSGi framework. It allows an Operator to configure deployed bundles. Configuring is the process of defining the configuration data for bundles and assuring that those bundles receive that data when they are active in the OSGi framework."
Configuration Admin Service: Targets

- A **Managed Service** represents a client of the Configuration Admin service, and is thus a configuration target. Bundles should register a Managed Service to receive the configuration data from the Configuration Admin service. A Managed Service adds one or more unique service.pid service properties as a primary key for the configuration information.

- A **Managed Service Factory** can receive a number of configuration dictionaries from the Configuration Admin service, and is thus also a configuration target service. (...)

_Not being used in the tutorial._
Configuration Admin Service Implementations

Export package org.osgi.service.cm

- Equinox
  org.eclipse.equinox.cm

- Felix
  org.apache.felix.configadmin
Service PID

The ManagedService should be registered with the Framework registry with the service.pid property set to some unique identifier. Default PID is the name of the Component. PID can be changed by using the configurationPid property of the @Component annotation.

@Component(
    ...,
    configurationPid = "my.unique.pid",
    ...
)


ConfigurationPolicy

Configuration Policy for the @Component annotation.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service.

- **OPTIONAL** (default)
- **REQUIRE**
- **(IGNORE)**

```java
@Component(
    configurationPolicy = ConfigurationPolicy.REQUIRE,
    ...
)
```
Metatype Service Specification Version 1.2

- The Metatype specification defines interfaces that allow bundle developers to describe attribute types in a computer readable form using so-called metadata.

- The purpose of the Metatype Service Specification is to allow services to specify the type information of data that they can use as arguments. The data is based on attributes, which are key/value pairs like properties.
Metatype Service Specification

- **Meta Type Service** provides a unified access point for meta type information.
- PID – A unique persistent ID, defined in configuration management.
- **Attribute Definition** defines a description, name, help text, and type information of an attribute.
- A **Object Class Definition** defines the type of a datum. It contains a description and name of the type plus a set of Attribute Definition objects.
Metatype Service Implementations

Export package org.osgi.service.metatype

- Equinox
  org.eclipse.equinox.metatype

- Felix
  org.apache.felix.metatype
Exemplary Metatype XML

Provided as <YOUR_BUNDLE>/OSGI-INF/metatype/foo.xml

<MetaData xmlns="...">

  <OCD description="My Foo Configuration Service" name="Foo Configuration" id="ocd">
    <AD name="Meals" id="meals" required="true" type="String" cardinality="2147483647" default="Pizza" description="Available meal that can be ordered and cooked." />
  </OCD>

  <Designate pid="com.eclipsesource.osgi.tutorial.KitchenWhiteboardImpl">
    <Object ocdref="ocd" />
  </Designate>

</MetaData>
Webconsole Configuration BEFORE
Webconsole Configuration AFTER

**Whiteboard Configuration**

- **Kitchen Whiteboard Configuration Service**
  - **Meals**
    - Burger
    - Salad
    - Pizza
  - Available meal that can be ordered and cooked. (meals)

**Configuration Information**

- Persistent Identity (PID) com.eclipsesource.osgi.tutorial.KitchenWhiteboardImpl

Buttons: Cancel, Reset, Delete, Unbind, Save
Add Resources to Bundles in BND

- Problem
  So far we’ve only added class files in Java packages (private or exported). But we need to add the metatype XML file to the bundle.

- Solution: Use `-includeresource` in `.bnd` file of a bundle
  Specify `to-location=from-location`

  `-includeresource: \ 
   location/in/bundle/foo.xml=project/location/bar.xml`
6.4 Add Types with MetatypeService

Metatype XML file already available in `resources/KitchenWhiteboardImpl.xml`. Include this XML file in bundle.

```
-includeresource: <osgi-INF/metatype/KitchenWhiteboardImpl.xml=resources/KitchenWhiteboardImpl.xml

(Hint: No editor support)```
Alternative to BND Tools

Builder in Eclipse that generates XML component files from annotations