

# BUILDING NANO SERVICES WITH OSGI

# Building Nano Services with OSGi Speaker



## **Dirk Fauth**

*Software-Architect Rich Client Systeme  
Eclipse Committer*

Robert Bosch GmbH  
Franz-Oechsle-Straße 4  
73207 Plochingen

[dirk.fauth@de.bosch.com](mailto:dirk.fauth@de.bosch.com)  
[www.bosch.com](http://www.bosch.com)  
[blog.vogella.com/author/fipro/](http://blog.vogella.com/author/fipro/)  
Twitter: [fipro78](#)

# Building Nano Services with OSGi Speaker



## **Peter Kirschner**

*Developer, Architect, Build and Release Engineer*

*OSS, OSGi & Eclipse Enthusiast*

Kirschners GmbH  
Löchgauer Straße 57  
74321 Bietigheim-Bissingen

[peter@kirschners.de](mailto:peter@kirschners.de)  
GitHub: [peterkir.github.io](https://github.com/peterkir)  
Twitter: [peterkir](https://twitter.com/peterkir)

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
5. Configuration
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# OVERVIEW

# Agenda

## 1. Overview

2. Tooling

3. Exercise: Simple Service

4. OSGi Console

5. Configuration

6. Exercise: Configurable Service

7. Felix OSGi Web Console

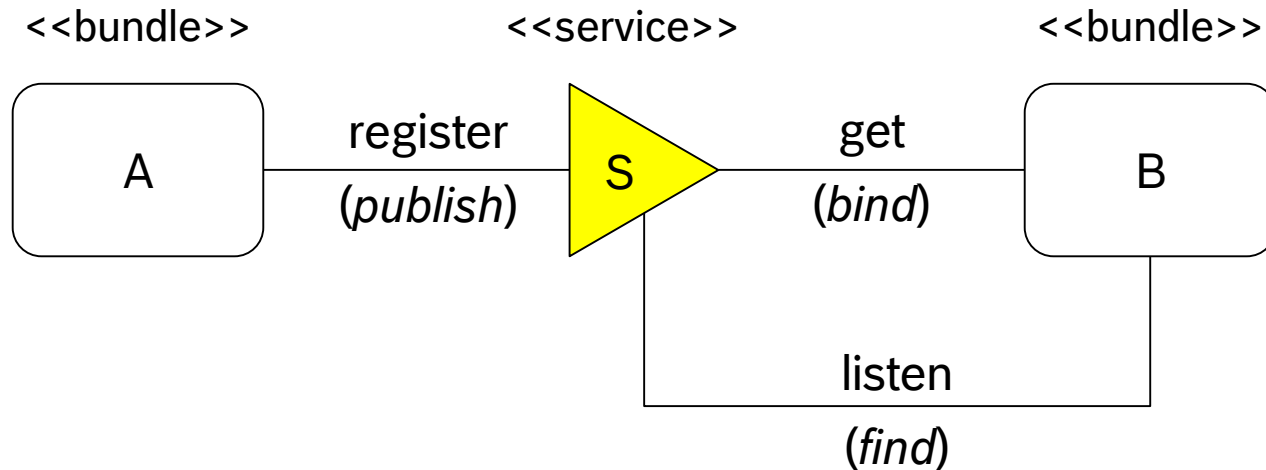
8. Remote Service Admin

9. Exercise: Remote Service

# Building Nano Services with OSGi

## Publish-Find-Bind

- ▶ Bundles **register (publish)** services
- ▶ Bundles **get (bind)** services
- ▶ Bundles **listen (find)** services



# Building Nano Services with OSGi Components

## ► (Service) Component

- Java class contained in a bundle
- Declared via *Component Description*

## ► Component Description

- XML document to declare a *Service Component*

## ► Component Configuration

- *Component Description* that is parameterized with component properties
- Tracks the component dependencies and manages the component instance

## ► Component Instance

- Instance of the component implementation class
- Created when a *Component Configuration* is activated
- Discarded when the *Component Configuration* is deactivated



# Building Nano Services with OSGi References

## ► References

- The definition of dependencies to other services.

## ► Target Services

- The services that match the reference interface and target property filter.

## ► Bound Services

- The services that are bound to a *Component Configuration*.

## ► Access Strategies

- Event Strategy
- Lookup Strategy
- Field Strategy (1.3)

# Building Nano Services with OSGi

## Component Types

### ► Delayed Component

- Activated when the service is requested
- Needs to specify a service

### ► Immediate Component

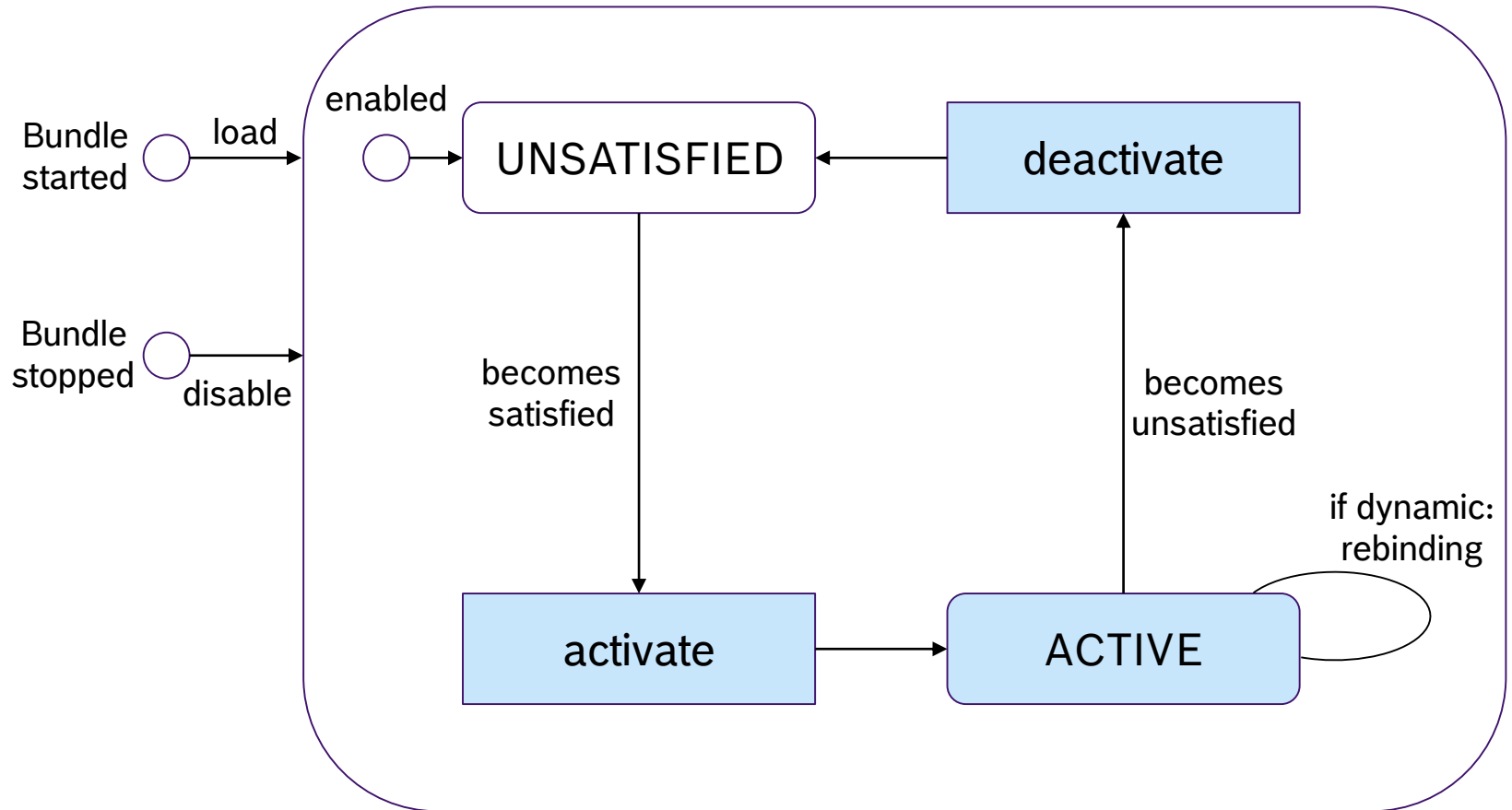
- Activated as soon as all dependencies are satisfied
- Does not need to specify a service

### ► Factory Component

- Creates and activates Component Configurations

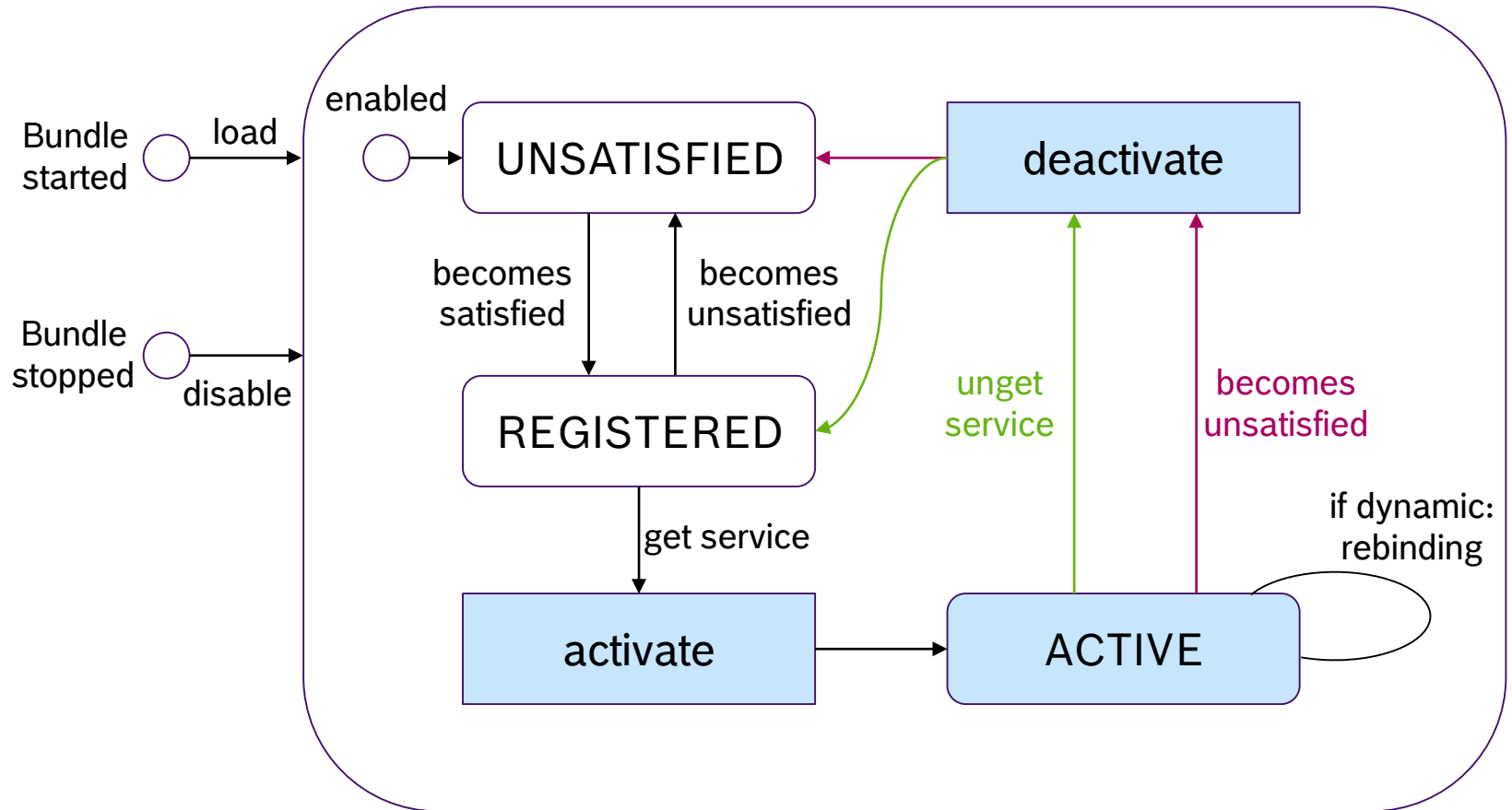
# Building Nano Services with OSGi

## Component Lifecycle – Immediate



# Building Nano Services with OSGi

## Component Lifecycle – Delayed



# Building Nano Services with OSGi

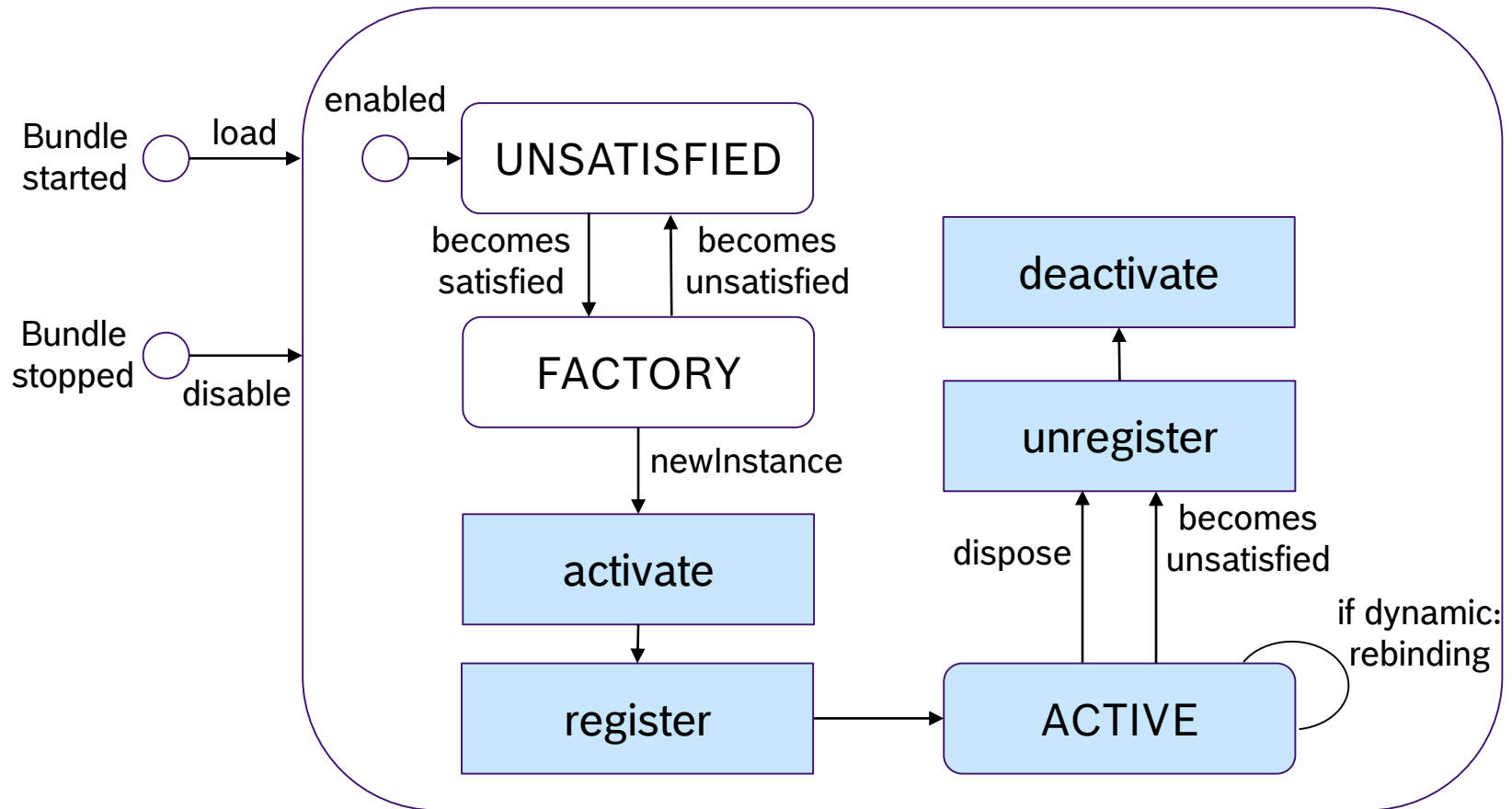
## Component Lifecycle – Activation

- ▶ Activation consists of the following steps:
  1. Load the component implementation class
  2. Create the component instance and component context
  3. Bind the target services
  4. Call the activate method if present
  
- ▶ For **Delayed Components** the load time is moved to the first request (including reference bindings)

*(see Declarative Services Specification Version 1.3 – 112.5.6 Activation)*

# Building Nano Services with OSGi

## Component Lifecycle – Factory



# TOOLING

# Agenda

1. Overview

**2. Tooling**

3. Exercise: Simple Service

4. OSGi Console

5. Configuration

6. Exercise: Configurable Service

7. Felix OSGi Web Console

8. Remote Service Admin

9. Exercise: Remote Service



# Building Nano Services with OSGi Tooling

## ► Since Eclipse Neon

- PDE - DS Annotations Support
- Contribution by Peter Nehrer (@pnehrer)



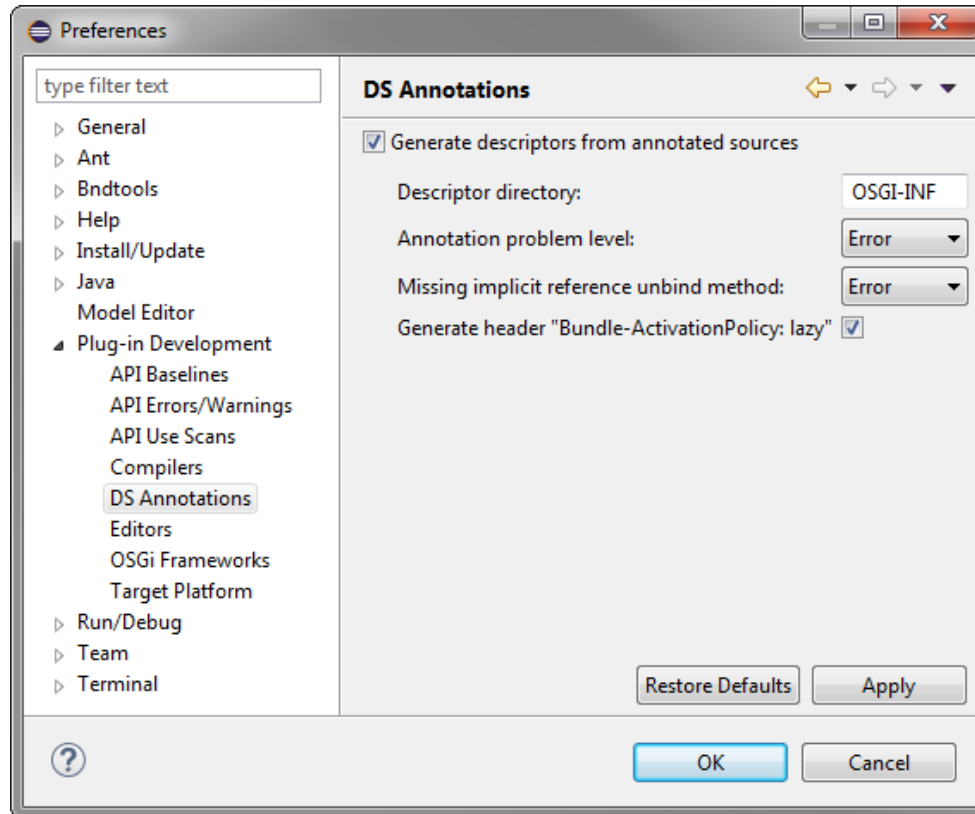
## ► Eclipse <= Mars

- Declarative Services Annotations Support ([Eclipse Marketplace](https://marketplace.eclipse.org/content/declarative-services-annotations-support))  
*<https://marketplace.eclipse.org/content/declarative-services-annotations-support>*

# Building Nano Services with OSGi

## PDE – DS Annotations Support

### ► Activate *DS Annotations Support* via *Preferences*



# Building Nano Services with OSGi

## Bndtools

- ▶ Additional Eclipse Plug-ins (bundles)
  - Focus on bnd configurations
  - OSGi metadata is generated
- ▶ Installation via Eclipse Marketplace or Update Site  
*<http://bndtools.org/installation.html>*



# Building Nano Services with OSGi

## PDE vs. Bndtools

Topic	PDE	Bndtools
Project layout	Special PDE project layout	Default Java project layout
MANIFEST.MF handling	Explicit editing	Generated out of meta-data
Package imports	Manually (manual triggered calculation)	Bytecode-based import calculation
Import/export package versions	Explicit properties	Implicit conventions
Launch defaults	Persisted, No Update	Clean, Hot Bundle Deploy
Bundle handling	IDE uses „virtual bundle“ Explicit export	Instant bundle creation
One Eclipse project	One bundle	Can become multiple bundles

# Building Nano Services with OSGi Provided Tooling

- ▶ IDEfix – Oomph Setup
  - Eclipse Committers Oxygen M2
  - Bndtools 3.3.0
- ▶ Added the following bundles to the installation to avoid p2 target definition
  - ECF Remote Services SDK 3.13.2
  - Equinox Target Components 3.12.0
  - Apache Felix Webconsole All 4.2.16

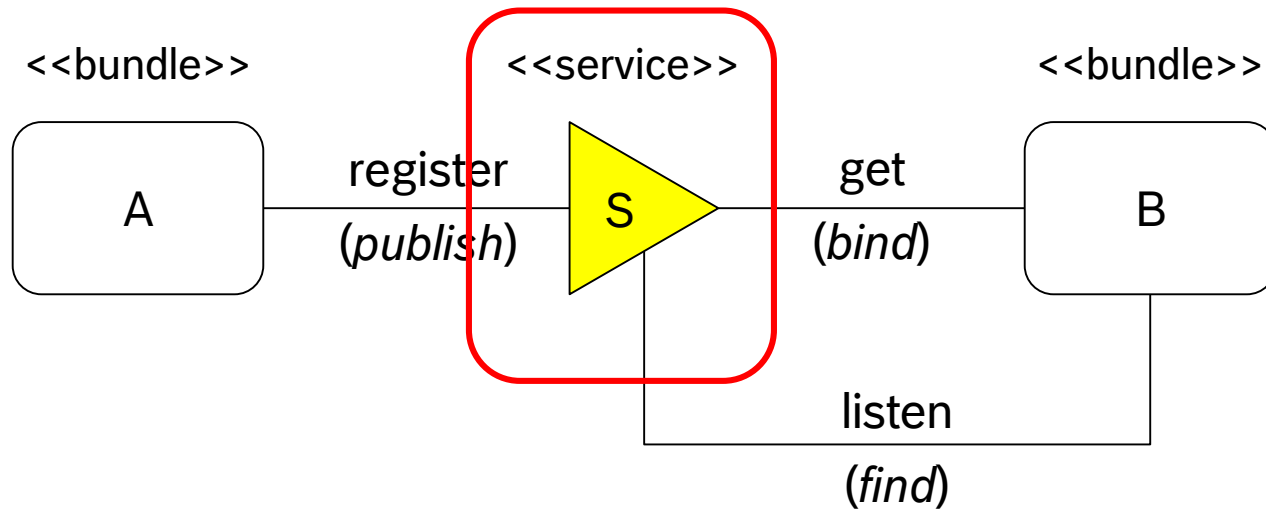
# EXERCISE: SIMPLE SERVICE

# Agenda

1. Overview
2. Tooling
- 3. Exercise: Simple Service**
  - 1. Service API**
  2. Service Implementation
  3. Service Consumer
  4. Launch in IDE
  5. Launch in standalone runtime
4. OSGi Console
5. Configuration
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi

## Service API





# Building Nano Services with OSGi

## Service API

- ▶ Create a bundle for the service API (e.g. *examples.service.api*)
- ▶ Create the service interface

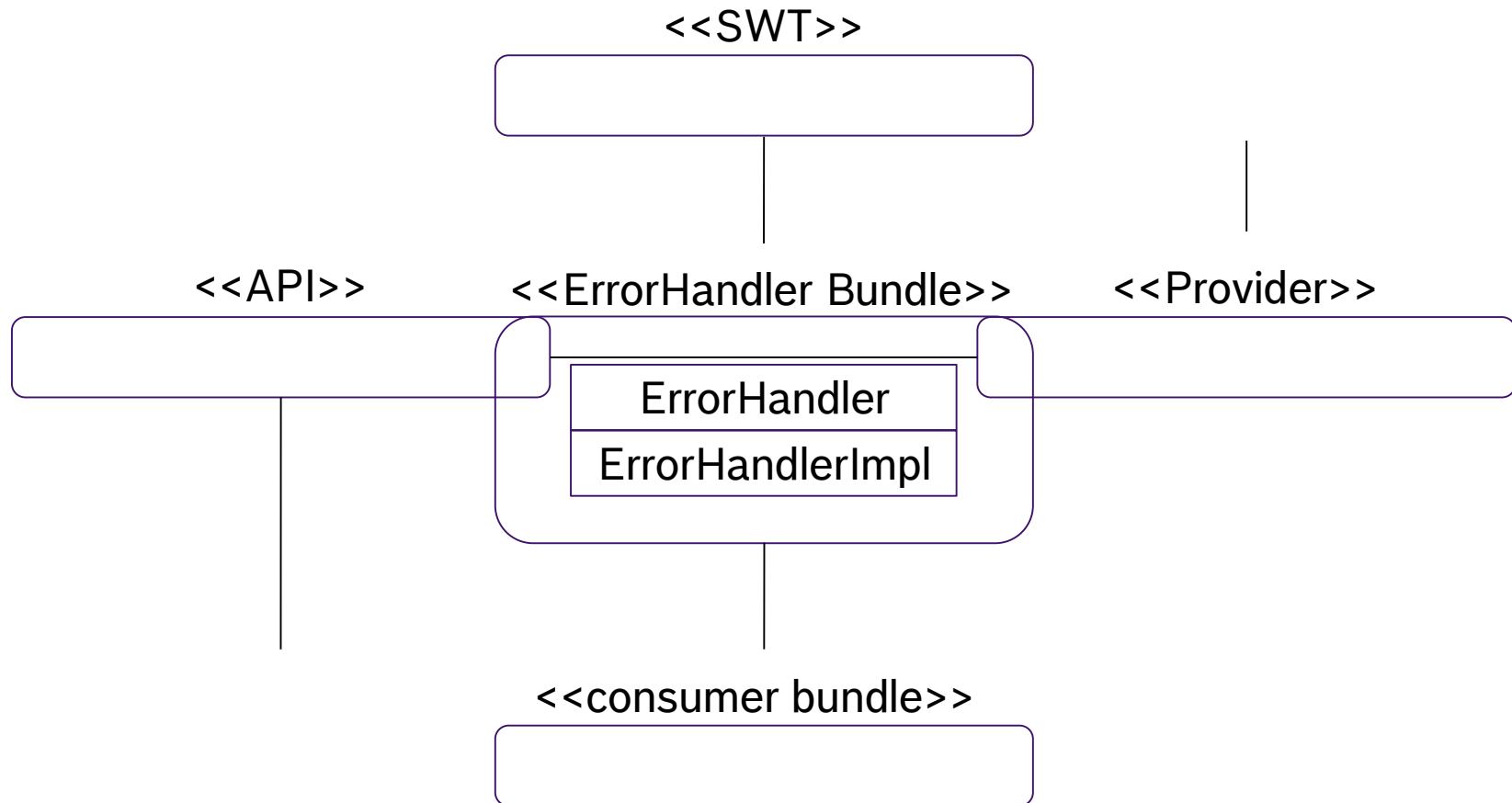
```
public interface StringModifier {  
    String modify(String input);  
}
```

**M**(ost) **U**(seless) **S**(ervice) **E**(ver)

- **Make the service implementation exchangeable**
- **Clean dependency hierarchy**

# Building Nano Services with OSGi

## Service API – Motivation

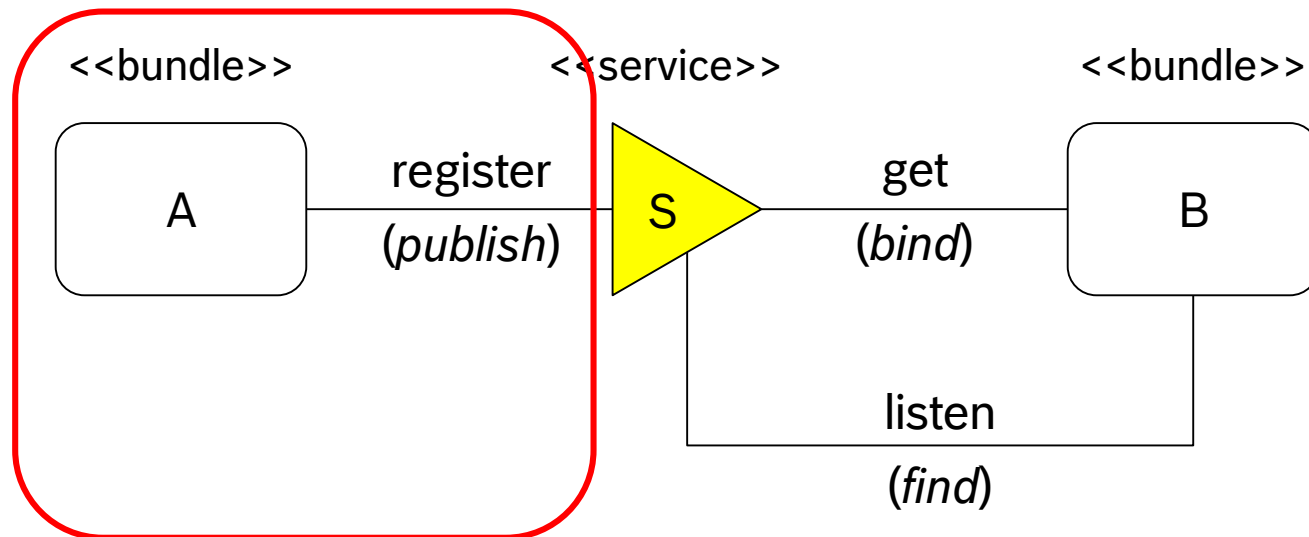


# Agenda

1. Overview
2. Tooling
- 3. Exercise: Simple Service**
  1. Service API
  - 2. Service Implementation**
  3. Service Consumer
  4. Launch in IDE
  5. Launch in standalone runtime
4. OSGi Console
5. Configuration
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi

## Service Implementation



# Building Nano Services with OSGi

## Service Implementation

- ▶ Create the *Service Component*
- ▶ Let the *Component Description* be generated via `@Component`



**@Component**

```
public class StringInverterImpl implements StringModifier {  
  
    @Override  
    public String modify(String input) {  
        return new StringBuilder(input).reverse().toString();  
    }  
}
```

# Building Nano Services with OSGi @Component

## ► Annotation to mark a Java class as a *Service Component*

## ► Generates (general)

- Generation of *Component Description* XML file
- Generation of `Service-Component` header in *MANIFEST.MF*

## ► Generates (PDE)

- `Bundle-ActivationPolicy: lazy` header in *MANIFEST.MF*
- Adds *Component Description* XML file to *build.properties*

## ► Generates (Bndtools)

- `Provide-Capability` header for `osgi.service`
- `Require-Capability` header for `osgi.extender` (*DS 1.3*)

# Building Nano Services with OSGi Capabilities

- ▶ Capability = non-code dependency
- ▶ `osgi.extender=osgi.component`

```
Require-Capability: osgi.extender;  
filter:="(&(osgi.extender=osgi.component) (version>=1.3) (! (version>=2.0)))"
```

- added to spec with DS 1.3
- Equinox DS adapted for DS 1.2 with Eclipse Neon

- ▶ `osgi.service`
  - specify the provided service implementations

```
Provide-Capability: osgi.service;  
objectClass:List<String>="org.fipro.inverter.StringInverter"
```

**The p2 resolver does not support OSGi capabilities!**

# Building Nano Services with OSGi

## @Component

Type Element	Default	Type Element	Default
<b>configurationPid</b>	full qualified class name of the component	<b>property</b>	empty
<b>configurationPolicy</b>	optional	<b>service</b>	full qualified class names of all <b>directly</b> implemented interfaces
<b>enabled</b>	true	<b>servicefactory</b> (depr. in 1.3)	false
<b>factory</b>	empty String	<b>xmlns</b>	lowest DS XML namespace that supports used features
<b>immediate</b>	false if <i>factory</i> or <i>service</i> true otherwise		
<b>name</b>	full qualified class name of the component	<b>reference</b> (since 1.3)	empty
<b>properties</b>	empty	<b>scope</b> (since 1.3)	SINGLETON



# Agenda

1. Overview

2. Tooling

## 3. Exercise: Simple Service

1. Service API

2. Service Implementation

### 3. Service Consumer

4. Launch in IDE

5. Launch in standalone runtime

4. OSGi Console

5. Configuration

6. Exercise: Configurable Service

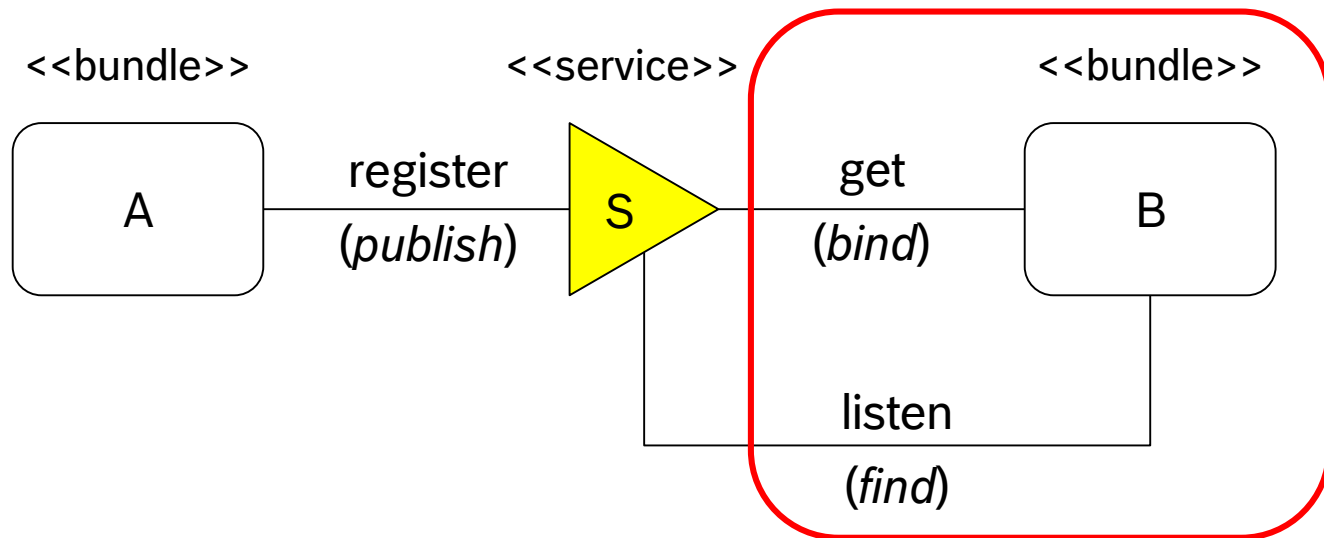
7. Felix OSGi Web Console

8. Remote Service Admin

9. Exercise: Remote Service

# Building Nano Services with OSGi


## Service Consumer



# Building Nano Services with OSGi

## Service Consumer

- ▶ Create the service consumer as *Service Component*
- ▶ Reference the *Service Component* that should be consumed via `@Reference`




```
@Component(property= {"osgi.command.scope:String=examples",
                      "osgi.command.function:String=modify"},
           service=StringModifierCommand.class
)
public class StringModifierCommand {

    private StringModifier modifier;

    @Reference
    void bindStringModifier(StringModifier modifier) { this.modifier = modifier; }

    public void modify(String input) { System.out.println(modifier.modify(input)); }
}
```



# Building Nano Services with OSGi @Reference

- ▶ **Specify dependency on other services**
- ▶ **Resolving references is required to satisfy a component**  
*(if the reference is not optional)*
- ▶ **Different strategies for accessing services**
  - *Event Strategy*
  - *Field Strategy (DS 1.3)*
  - *Lookup Strategy*

# Building Nano Services with OSGi

## @Reference – Event Strategy

- **Event Strategy** – using event methods for *bind/updated/unbind*

```
@Component(...)
public class StringModifierCommand {

    private StringModifier modifier;

    @Reference
    void bindStringModifier(StringModifier modifier) { this.modifier = modifier; }

    void updatedStringModifier(
        StringModifier modifier, Map<String, ?> properties) { //do something }

    void unbindStringModifier(StringModifier modifier) { this.modifier = null; }

    public void modify(String input) { System.out.println(modifier.modify(input)); }
}
```

# Building Nano Services with OSGi

## @Reference - Event Method Parameter

### ► ServiceReference

```
void bindStringModifier(ServiceReference<StringModifier> reference)
```

### ► <service type>

```
void bindStringModifier(StringModifier modifier)
```

### ► <service type> + Map<String, ?>

```
void bindStringModifier(  
    StringModifier modifier, Map<String, Object> properties)
```

## With DS 1.3

### ► ComponentServiceObjects

### ► Different variations of the parameter list

# Building Nano Services with OSGi

## @Reference – Field Strategy

### ► **Field Strategy** (DS 1.3) – using instance fields

```
@Component(...)
public class StringModifierCommand {

    @Reference
    private StringModifier modifier;

    public void modify(String input) {
        System.out.println(modifier.modify(input));
    }
}
```

# Building Nano Services with OSGi

## @Reference – Lookup Strategy (DS 1.2)

- **Lookup Strategy** (DS 1.2) – lookup everytime needed, do not store

```
@Component(...)
public class StringModifierCommand {

    private ComponentContext context;
    private ServiceReference<StringModifier> reference;

    @Activate
    void activate(ComponentContext context) { this.context = context; }

    @Reference
    void setStringModifier(ServiceReference<StringModifier> reference) { this.reference = reference; }

    public void modify(String input) {
        StringModifier modifier =
            (StringModifier) context.locateService("StringModifier", reference);

        ...
    }
}
```



# Building Nano Services with OSGi

## @Reference – Lookup Strategy (DS 1.3)

- **Lookup Strategy** (DS 1.3) – lookup everytime needed, do not store

```
@Component(...
    reference=@Reference(name="modifier", service=StringModifier.class)
)
public class StringModifierCommand {

    private ComponentContext context;

    @Activate
    void activate(ComponentContext context) { this.context = context; }

    public void modify(String input) {
        StringModifier modifier = (StringModifier) context.locateService("modifier");
        ...
    }
}
```

# Building Nano Services with OSGi

## @Reference

Type Element	Default	Type Element	Default
<b>cardinality</b>	<ul style="list-style-type: none"> <li>1:1 for event methods</li> <li>1:1 for non-collection fields, 0..n for collections</li> </ul>	<b>unbind</b>	unbind<name> unset<name> remove<name>
<b>name</b>	<ul style="list-style-type: none"> <li>bind event method name without bind prefix</li> <li>name of the field</li> </ul>	<b>updated</b>	updated<name> if such a method exists
<b>policy</b>	STATIC		
<b>policyOption</b>	RELUCTANT	<b>bind</b> (since 1.3)	the name of the annotated method or empty
<b>service</b>	full qualified class name of the referenced service	<b>field</b> (since 1.3)	the name of the annotated field or empty
<b>target</b>	empty String	<b>fieldOption</b> (since 1.3)	REPLACE
		<b>scope</b> (since 1.3)	BUNDLE

# Agenda

1. Overview

2. Tooling

## 3. Exercise: Simple Service

1. Service API

2. Service Implementation

3. Service Consumer

## 4. Launch in IDE

## 5. Launch in standalone runtime

4. OSGi Console

5. Configuration

6. Exercise: Configurable Service

7. Felix OSGi Web Console

8. Remote Service Admin

9. Exercise: Remote Service

# Building Nano Services with OSGi

## Launch in IDE / standalone runtime

### ► Required bundles

- *examples.pde.command*
- *examples.pde.impl.inverter*
- *examples.pde.service.api*



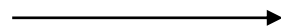
Application Bundles

- *org.apache.felix.gogo.command*
- *org.apache.felix.gogo.runtime*
- *org.apache.felix.gogo.shell*
- *org.eclipse.equinox.console*



OSGi Console

- *org.eclipse.equinox.ds*
- *org.eclipse.equinox.util*
- *org.eclipse.osgi*
- *org.eclipse.osgi.services*



Equinox DS Impl



Equinox Utils



Equinox OSGi Impl



Equinox OSGi Service Interfaces

# OSGI CONSOLE (GOGO SHELL)

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
- 4. OSGi Console**
  - 1.Available Commands**
  - 2.Interlude: Lifecycle Methods
  - 3.Exercise
5. Configuration
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi

## OSGi Console – Commands

Function	Equinox Console	Felix Gogo Shell
<b>Framework Commands</b>		
List installed bundles	<code>ss   lb [bundle-id   bundle-name]</code>	<code>lb [bundle-id   bundle-name]</code>
Start a bundle	<code>start &lt;bundle-id&gt;</code>	<code>start &lt;bundle-id&gt;</code>
Stop a bundle	<code>stop &lt;bundle-id&gt;</code>	<code>stop &lt;bundle-id&gt;</code>
List all available commands	<code>help [sub-command]</code>	<code>help [sub-command]</code>
Exit the runtime	<code>exit</code>	<code>exit 0</code>
<b>SCR Commands</b>		
List all components	<code>list   ls [bundle-id]</code>	<code>scr:list [bundle-id]</code>
Print all component information	<code>component   comp &lt;component-id&gt;</code>	<code>scr:info &lt;component-id&gt;</code>
Enable a component	<code>enable   en &lt;component-id&gt;</code>	<code>scr:enable &lt;component-name&gt;</code>
Disable a component	<code>disable   dis &lt;component-id&gt;</code>	<code>scr:disable &lt;component-name&gt;</code>

<xxx> = mandatory parameter

[xxx] = optional parameter

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
- 4. OSGi Console**
  1. Available Commands
  - 2. Interlude: Lifecycle Methods**
  3. Exercise
5. Configuration
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service



# Building Nano Services with OSGi

## @Activate / @Modified / @Deactivate

### ► Component life cycle methods

### ► Method parameters

- ComponentContext
- BundleContext
- Map<String, ?>  
Map containing component properties
- int / Integer for (@Deactivate)
- <Component Property Type> (DS 1.3)  
Type safe access to component properties

```
@Activate
private void activate(
    ComponentContext c,
    BundleContext b,
    Map<String, ?> properties) {

    //do some initialization stuff
}
```

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
- 4. OSGi Console**
  1. Available Commands
  2. Interlude: Lifecycle Methods
  - 3. Exercise**
5. Configuration
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# CONFIGURATION

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
- 5. Configuration**
  - 1.Component Properties**
  - 2.Configuration Admin Service
  - 3.Configuration Change Notifications
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi

## Component Properties

- ▶ Set of properties → key-value-pairs
- ▶ Can be specified via
  - **Component Description** (directly vs. Java Properties file)  
Static configuration of default values or for usage by referencing components
  - **Configuration Admin Service**  
Dynamic configuration
  - **Component Factory**  
Dynamic configuration per created *Component Instance*
- ▶ Can be consumed
  - in lifecycle methods (*Component Properties*)
  - in event methods (*Service Properties*)

# Building Nano Services with OSGi

## Component Properties – Definition

### ► Direct

- `<name>(:<type>)?=<value>`
- Type is optional, defaults to String

```
@Component (  
    property = {  
        "osgi.command.scope:String=examples",  
        "osgi.command.function:String=modify"  
    },  
    service = StringModifierCommand.class)  
public class StringModifierCommand {
```

### ► Java Properties file

- Only values of type String
- Properties file needs to be located in the bundle

```
@Component (  
    properties = OSGI-INF/config.properties,  
    service = StringModifierCommand.class)  
public class StringModifierCommand {
```

# Building Nano Services with OSGi

## Component Properties – Consume DS 1.2

- *Via Component Properties* `Map<String, Object>` in lifecycle methods

```
@Activate
private void activate(Map<String, Object> properties) {
    // do some initialization stuff
}
```

- *Via Service Properties* `Map<String, Object>` in event methods

```
@Reference
void bindModifier(StringModifier modifier, Map<String, Object> properties) {
    // check properties
    this.modifier = modifier;
}
```

# Building Nano Services with OSGi

## Component Properties – Consume DS 1.3

### ► Component Property Types

- Type safe access to *Component Properties*
- Only usable in lifecycle methods
- Implemented as annotation (although not used as such)
  - no-argument methods
  - limited return types supported
  - support of default values
- Can be combined with `Map<String, Object>`

```
@interface Config {  
    String message() default "";  
    int iteration() default 0;  
}
```

```
@Activate  
private void activate(Config config) {  
    // do some initialization stuff  
}
```



# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
- 5. Configuration**
  1. Component Properties
  - 2. Configuration Admin Service**
  3. Configuration Change Notifications
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi Configuration Admin Service

## ► *Component Configuration* attributes

- `configurationPid`

PID (Persistent IDentity) = key for components that need a Configuration object  
default: PID → Component Name → fully qualified class name

- `configurationPolicy`

Relationship between *Component* and Configuration object

- OPTIONAL      – use if available, but be satisfied even without (default)
- REQUIRE       – Component can only be satisfied if a Configuration object is available
- IGNORE        – Never use an available Configuration object

# Building Nano Services with OSGi

## Configuration Admin Service - Configuration

### ► Get a Configuration object via

`ConfigurationAdmin#getConfiguration(String)`

- get an existing object from the persistent store or creates a new one

### ► Check if the Configuration already contains properties

- If yes, use them to ensure all configuration values are set
- If not, create a new `Dictionary`

### ► Update the properties of the Configuration

```
Configuration config = cm.getConfiguration("PID");
Dictionary<String, Object> props = null;
if (config != null && config.getProperties() != null) {
    props = config.getProperties();
} else {
    props = new Hashtable<>();
}
...
config.update(props);
```

# Building Nano Services with OSGi

## Configuration Admin Service – Location Binding

- ▶ `Configuration` objects can be bound to a bundle
  - Bundles can't access `Configuration` objects that are bound to another bundle
- ▶ Creation of the `Configuration` object can influence the binding
  - `ConfigurationAdmin#getConfiguration(<PID>)`  
Configuration object is bound to the calling bundle
  - `ConfigurationAdmin#getConfiguration(<PID>, null)`  
Configuration object is bound to the first requestor of the `Configuration`
  - `ConfigurationAdmin#getConfiguration(<PID>, "?")`  
Multi-location-binding – Configuration object is bound to all targets with the matching PID

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
- 5. Configuration**
  1. Component Properties
  2. Configuration Admin Service
  - 3. Configuration Change Notifications**
6. Exercise: Configurable Service
7. Felix OSGi Web Console
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi

## Configuration Change Notifications

- Inside the configurable component – `@Modified` lifecycle method

```
@Modified
void modified(Map<String, Object> properties) {
    // do something with the change properties
}
```

- Inside the referencing component – `updated` event method

```
void updatedStringModifier(
    StringModifier modifier, Map<String, Object> properties) {
    // do something with the reference and properties
}
```

# EXERCISE: CONFIGURABLE SERVICE

# Building Nano Services with OSGi

## Launch in IDE / standalone runtime

### ► Additional required bundles:

- *examples.pde.impl.configurable* —→ Application Bundle
- *org.eclipse.equinox.cm* —→ Equinox Configuration Admin Service
- *org.eclipse.equinox.metatype* —→ Equinox Metatype Service



# FELIX WEBCONSOLE

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
5. Configuration
6. Exercise: Configurable Service

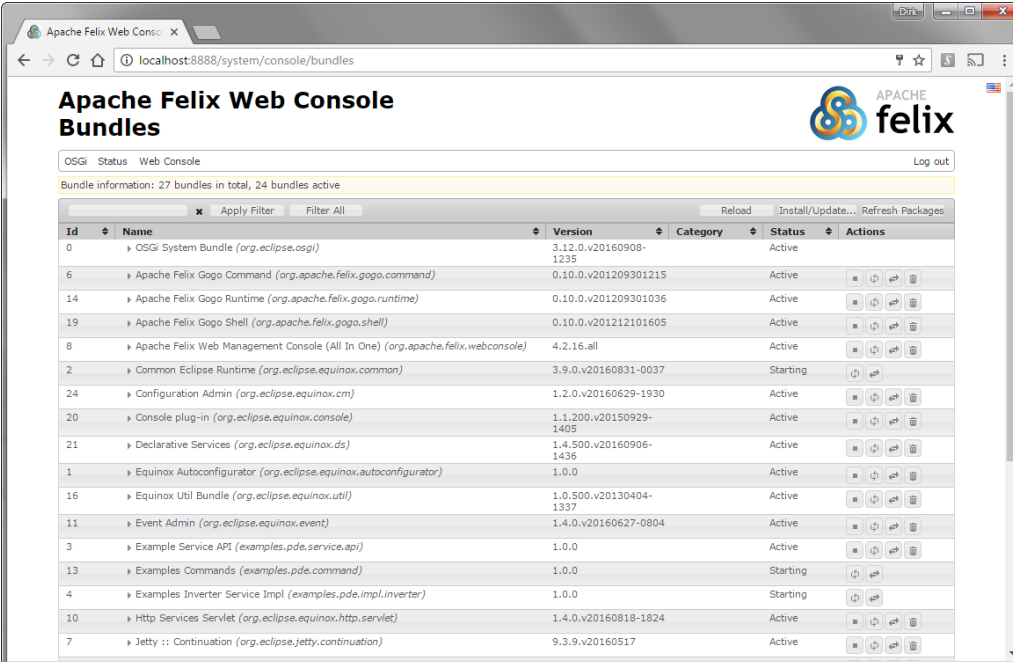
## 7. Felix OSGi Web Console

1. Introduction
  2. Interlude: Metatype
  3. Launch in IDE
  4. Launch in standalone runtime
8. Remote Service Admin
  9. Exercise: Remote Service

# Building Nano Services with OSGi

## Felix Webconsole – Introduction

- ▶ Tool to **inspect** and **manage** OSGi framework instances using a browser
- ▶ Requires a running OSGi Http Service implementation



The screenshot shows the Apache Felix Web Console interface in a web browser. The browser address bar shows the URL `localhost:8888/system/console/bundles`. The page title is "Apache Felix Web Console Bundles". Below the title, there is a "Log out" link. A status bar indicates "Bundle information: 27 bundles in total, 24 bundles active". Below this, there is a table of bundles with columns: Id, Name, Version, Category, Status, and Actions. The table lists various bundles including OSGi System Bundle, Apache Felix Gogo Command, Apache Felix Gogo Runtime, Apache Felix Gogo Shell, Apache Felix Web Management Console, Common Eclipse Runtime, Configuration Admin, Console plug-in, Declarative Services, Equinox Autoconfigurator, Equinox Util Bundle, Event Admin, Example Service API, Examples Commands, Examples Inverter Service Impl, Http Services Servlet, and Jetty :: Continuation.

Id	Name	Version	Category	Status	Actions
0	OSGi System Bundle ( <i>org.eclipse.osgi</i> )	3.12.0.v20160908-1235		Active	
6	Apache Felix Gogo Command ( <i>org.apache.felix.gogo.command</i> )	0.10.0.v201209301215		Active	
14	Apache Felix Gogo Runtime ( <i>org.apache.felix.gogo.runtime</i> )	0.10.0.v201209301036		Active	
19	Apache Felix Gogo Shell ( <i>org.apache.felix.gogo.shell</i> )	0.10.0.v201212101605		Active	
8	Apache Felix Web Management Console (All In One) ( <i>org.apache.felix.webconsole</i> )	4.2.16.all		Active	
2	Common Eclipse Runtime ( <i>org.eclipse.equinox.common</i> )	3.9.0.v20160831-0037		Starting	
24	Configuration Admin ( <i>org.eclipse.equinox.cm</i> )	1.2.0.v20160629-1930		Active	
20	Console plug-in ( <i>org.eclipse.equinox.console</i> )	1.1.200.v20150929-1405		Active	
21	Declarative Services ( <i>org.eclipse.equinox.ds</i> )	1.4.500.v20160906-1436		Active	
1	Equinox Autoconfigurator ( <i>org.eclipse.equinox.autoconfigurator</i> )	1.0.0		Active	
16	Equinox Util Bundle ( <i>org.eclipse.equinox.util</i> )	1.0.500.v20130404-1337		Active	
11	Event Admin ( <i>org.eclipse.equinox.event</i> )	1.4.0.v20160627-0804		Active	
3	Example Service API ( <i>examples.pde.service.api</i> )	1.0.0		Active	
13	Examples Commands ( <i>examples.pde.command</i> )	1.0.0		Starting	
4	Examples Inverter Service Impl ( <i>examples.pde.impl.inverter</i> )	1.0.0		Starting	
10	Http Services Servlet ( <i>org.eclipse.equinox.http.servlet</i> )	1.4.0.v20160818-1824		Active	
7	Jetty :: Continuation ( <i>org.eclipse.jetty.continuation</i> )	9.3.9.v20160517		Active	

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
5. Configuration
6. Exercise: Configurable Service

## 7. Felix OSGi Web Console

1. Introduction
  - 2. Interlude: Metatype**
  3. Launch in IDE
  4. Launch in standalone runtime
8. Remote Service Admin
  9. Exercise: Remote Service

# Building Nano Services with OSGi

## OSGi Metatype

- ▶ Dynamic typing system for *Configuration Properties*
- ▶ Purpose: dynamic construction of reasonable User Interfaces
- ▶ DS 1.2 – Manual creation of metatype XML file in *OSGI-INF/metatype*  
[Example](#)
- ▶ DS 1.3 – Generation of metatype XML file in combination with *Component Property Type*

```
@ObjectClassDefinition  
  
public @interface ModifierConfig {  
    String prefix() default "";  
    String suffix() default "";  
    int iteration() default 0;  
    boolean uppercase default false;  
}
```

# Agenda

1. Overview
2. Tooling
3. Exercise: Simple Service
4. OSGi Console
5. Configuration
6. Exercise: Configurable Service

## 7. Felix OSGi Web Console

1. Introduction
2. Interlude: Metatype
- 3. Launch in IDE**
- 4. Launch in standalone runtime**
8. Remote Service Admin
9. Exercise: Remote Service

# Building Nano Services with OSGi

## Launch in IDE / standalone runtime

### ► Additional required bundles:

- *javax.servlet* → Servlet API
- *org.apache.felix.webconsole* → Felix WebConsole (all)
- *org.eclipse.equinox.http.jetty*  
• *org.eclipse.equinox.http.servlet* } Equinox OSGi Http Service
- *org.eclipse.jetty.continuation*  
• *org.eclipse.jetty.http*  
• *org.eclipse.jetty.io*  
• *org.eclipse.jetty.security*  
• *org.eclipse.jetty.server*  
• *org.eclipse.jetty.servlet*  
• *org.eclipse.jetty.util* } Eclipse Jetty

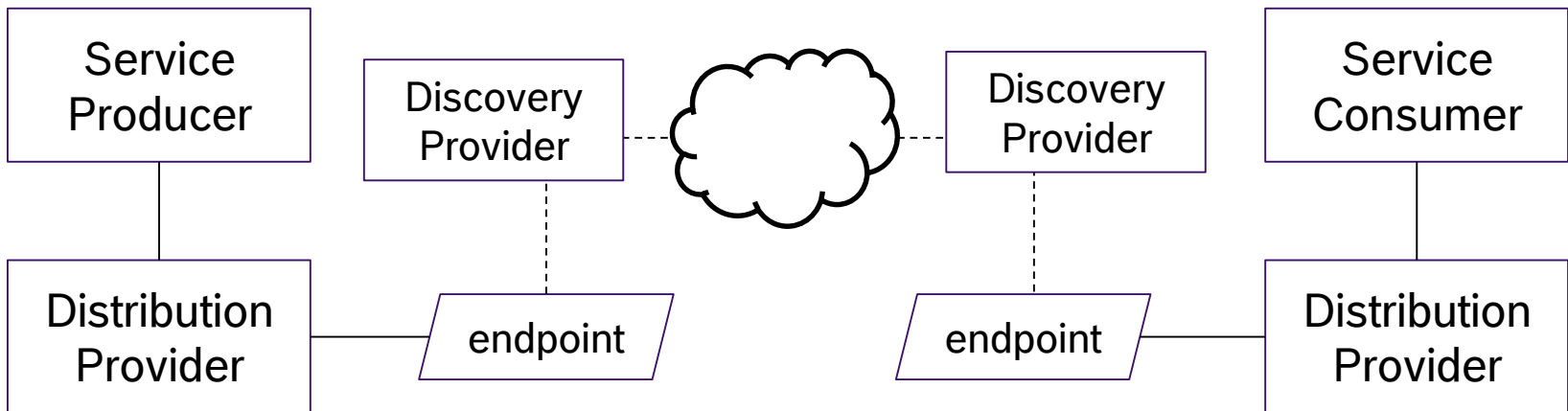
# REMOTE SERVICE ADMIN



# Building Nano Services with OSGi

## Remote Service Admin - Overview

- ▶ Export an OSGi service as a remote service
- ▶ Discovery Provider
  - Publish/Consume endpoints
- ▶ Distribution Provider
  - Export/Import services via endpoints



# Building Nano Services with OSGi

## Remote Service Admin – Service Properties

### ► `service.exported.interfaces=*`

Required property to specify which service interfaces should be exported. Using the wildcard it says that all the interfaces that are registered should be exported.

### ► `service.exported.configs`

The ECF container type/container factory name for the desired provider. The ECF project provides several distribution providers, e.g.

- ECF Generic Provider = `ecf.generic.server`
- ECF r-OSGi Provider = `ecf.r_osgi.peer`
- and several more

[https://wiki.eclipse.org/Distribution\\_Providers](https://wiki.eclipse.org/Distribution_Providers)

# EXERCISE: REMOTE SERVICE

# Building Nano Services with OSGi

## Launch in IDE / standalone runtime

### ► Additional required bundles:

- ***javax.servlet*** → Servlet API
- ***org.eclipse.core.jobs*** } Eclipse / Equinox
- ***org.eclipse.equinox.common*** } Concurrency
- ***org.eclipse.equinox.concurrent*** }
- ***org.eclipse.equinox.event*** → Equinox Event Admin
- ***org.eclipse.ecf*** }
- ***org.eclipse.ecf.discovery*** }
- ***org.eclipse.ecf.identity*** }
- ***org.eclipse.ecf.osgi.services.distribution*** }
- ***org.eclipse.ecf.osgi.services.remoteserviceadmin*** }
- ***org.eclipse.ecf.osgi.services.remoteserviceadmin.proxy*** }
- ***org.eclipse.ecf.provider*** }
- ***org.eclipse.ecf.provider.jmdns*** }
- ***org.eclipse.ecf.provider.remoteservice*** }
- ***org.eclipse.ecf.remoteservice*** }
- ***org.eclipse.ecf.remoteservice.asyncproxy*** }
- ***org.eclipse.ecf.sharedobject*** }
- ***org.eclipse.osgi.services.remoteserviceadmin*** }

# Building Nano Services with OSGi

## Further information

- ▶ OSGi Compendium Specification

*<https://www.osgi.org/developer/specifications/>*

- ▶ enRoute Documentation

*<http://enroute.osgi.org/book/210-doc.html>*

- ▶ Blog posts

- Getting Started with OSGi Declarative Services

- <http://blog.vogella.com/2016/06/21/getting-started-with-osgi-declarative-services/>*

- OSGi Component Testing

- <http://blog.vogella.com/2016/07/04/osgi-component-testing/>*

- Configuring OSGi Declarative Services

- <http://blog.vogella.com/2016/09/26/configuring-osgi-declarative-services/>*



Evaluate the Sessions

Sign in and vote at **[eclipsecon.org](http://eclipsecon.org)**

- 1 0 + 1