OSGI DECLARATIVE SERVICES

Getting Started with OSGi Declarative Services
Speaker

Dirk Fauth
Software-Architect Rich Client Systeme
Eclipse Committer

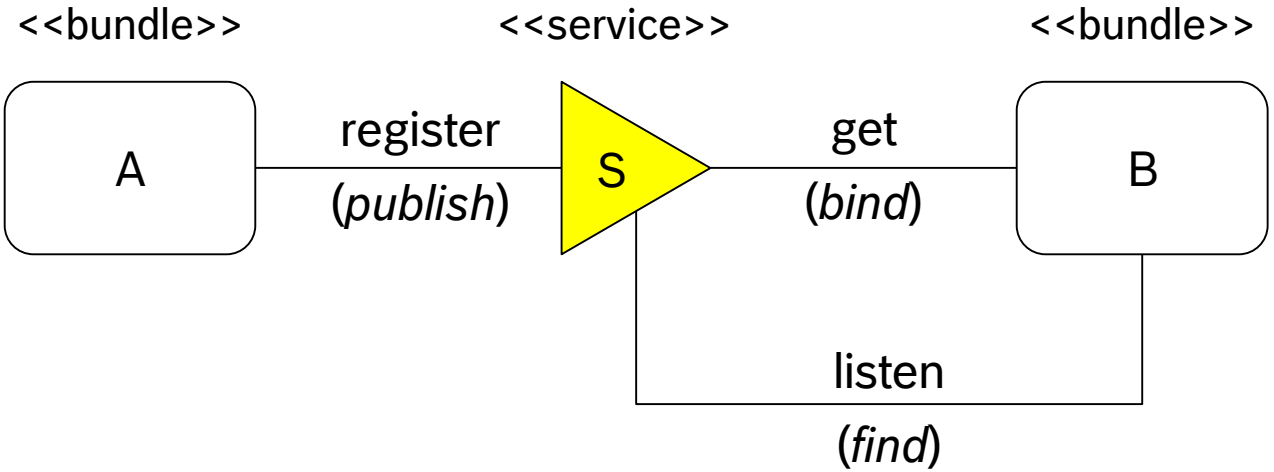
Robert Bosch GmbH
Franz-Oechsle-Straße 4
73207 Plochingen

dirk.fauth@de.bosch.com
www.bosch.com
blog.vogella.com/author/fipro/
Twitter: fipro78

OVERVIEW

OSGi Declarative Services
Publish-Find-Bind

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



OSGi Declarative Services
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

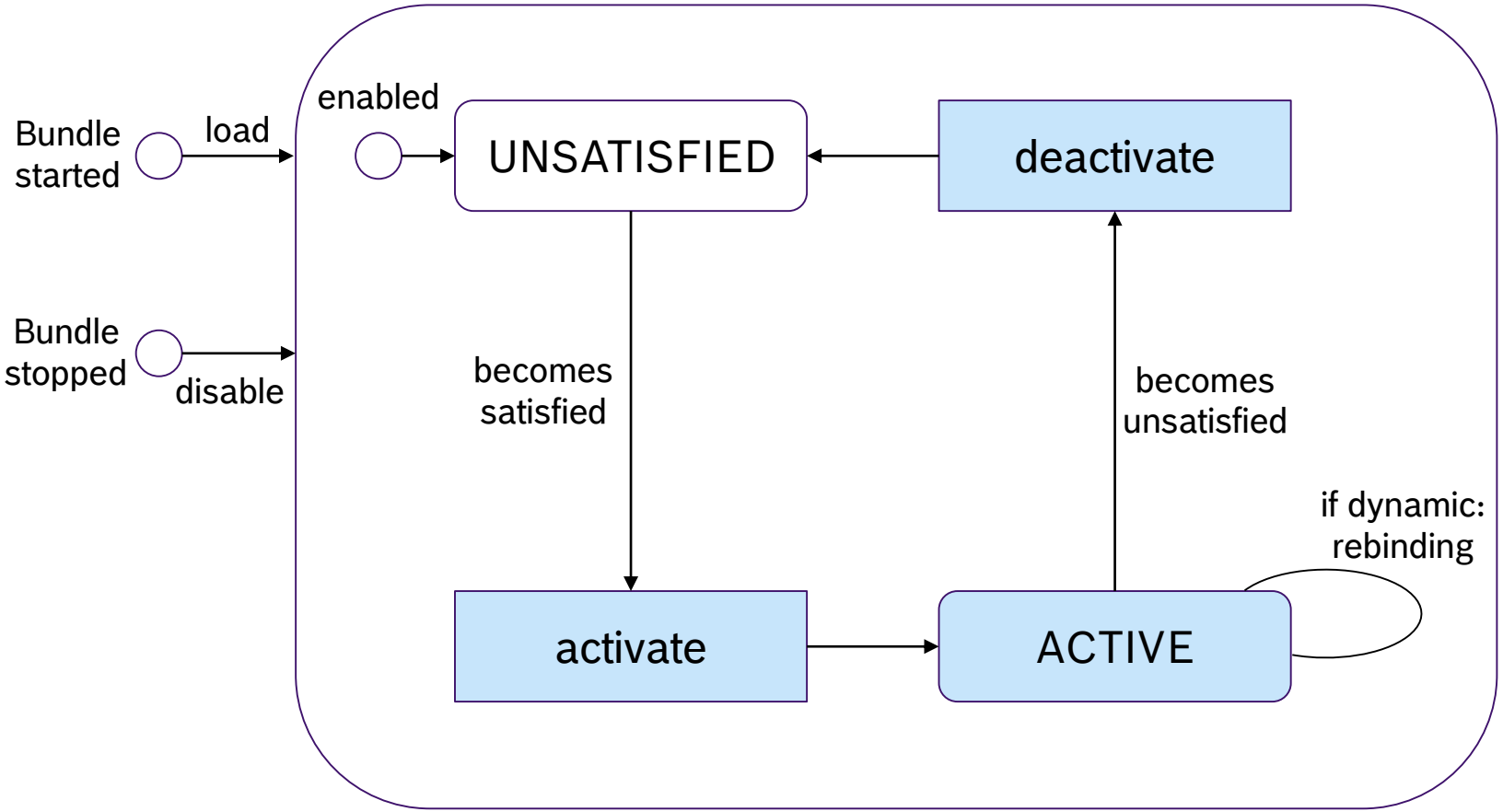
OSGi Declarative Services
References

- **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)

OSGi Declarative Services
Component Types

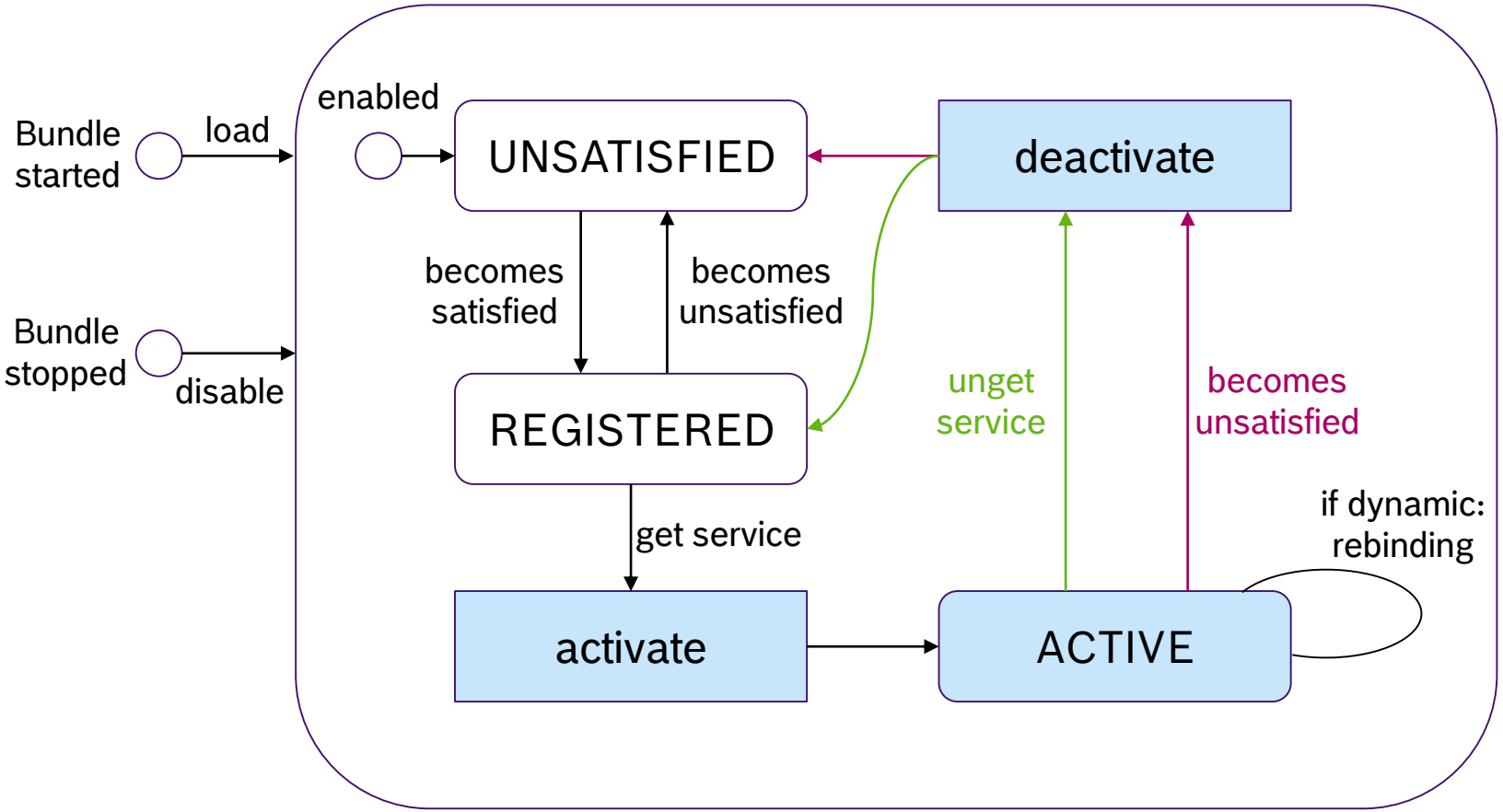
- **Delayed Component**
  - Activated when the service is requested
  - Needs to specify a service
- **Immediate Component**
  - Activated as soon as all dependencies are satisfied
  - Does not need to specify a service
- **Factory Component**
  - Creates and activates Component Configurations

OSGi Declarative Services
Component Lifecycle – Immediate



BOSCH

OSGi Declarative Services
Component Lifecycle – Delayed



BOSCH

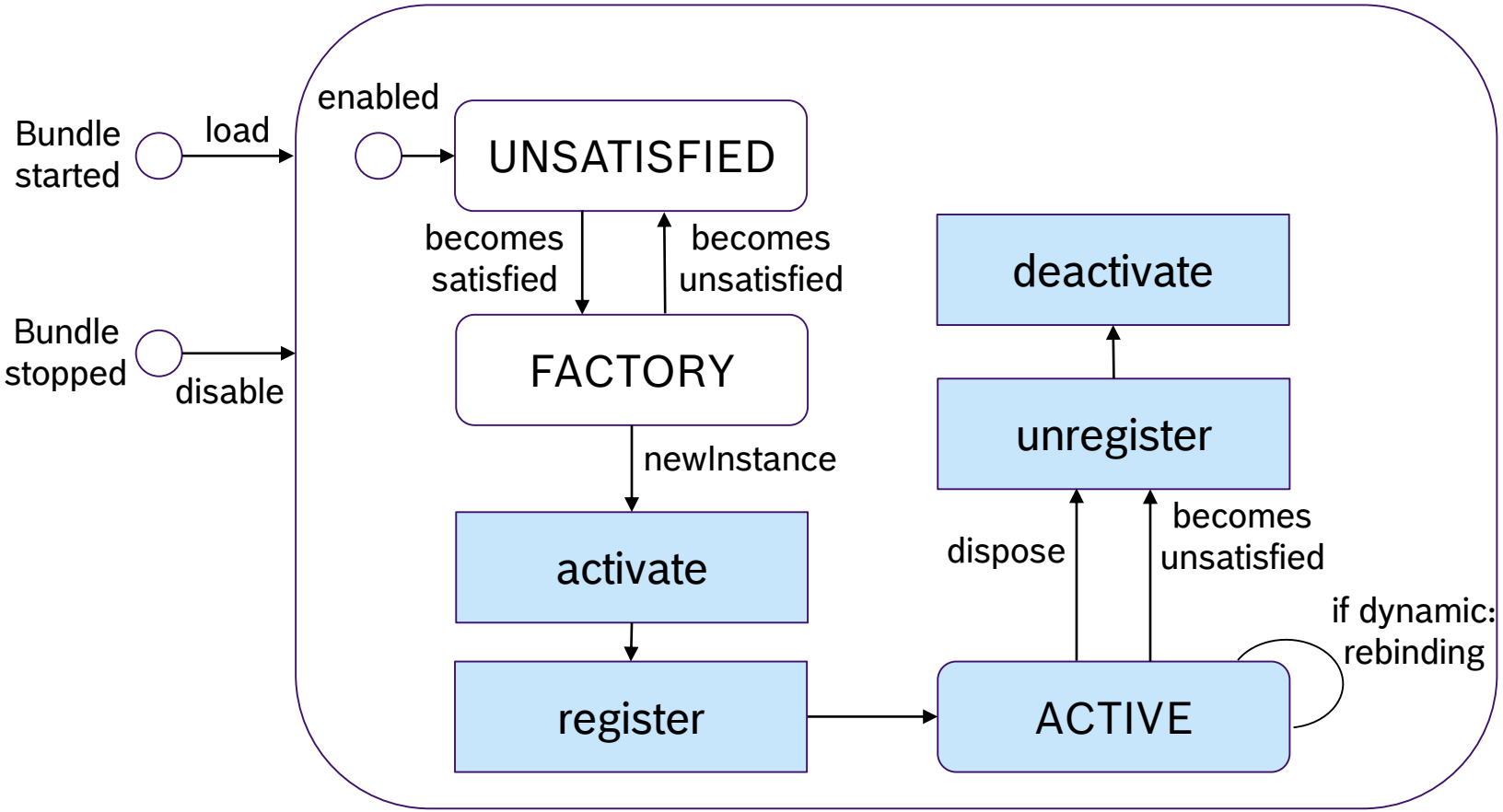
OSGi Declarative Services
Component Lifecycle – Activation

- 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)

OSGi Declarative Services
Component Lifecycle – Factory



BOSCH

TOOLING

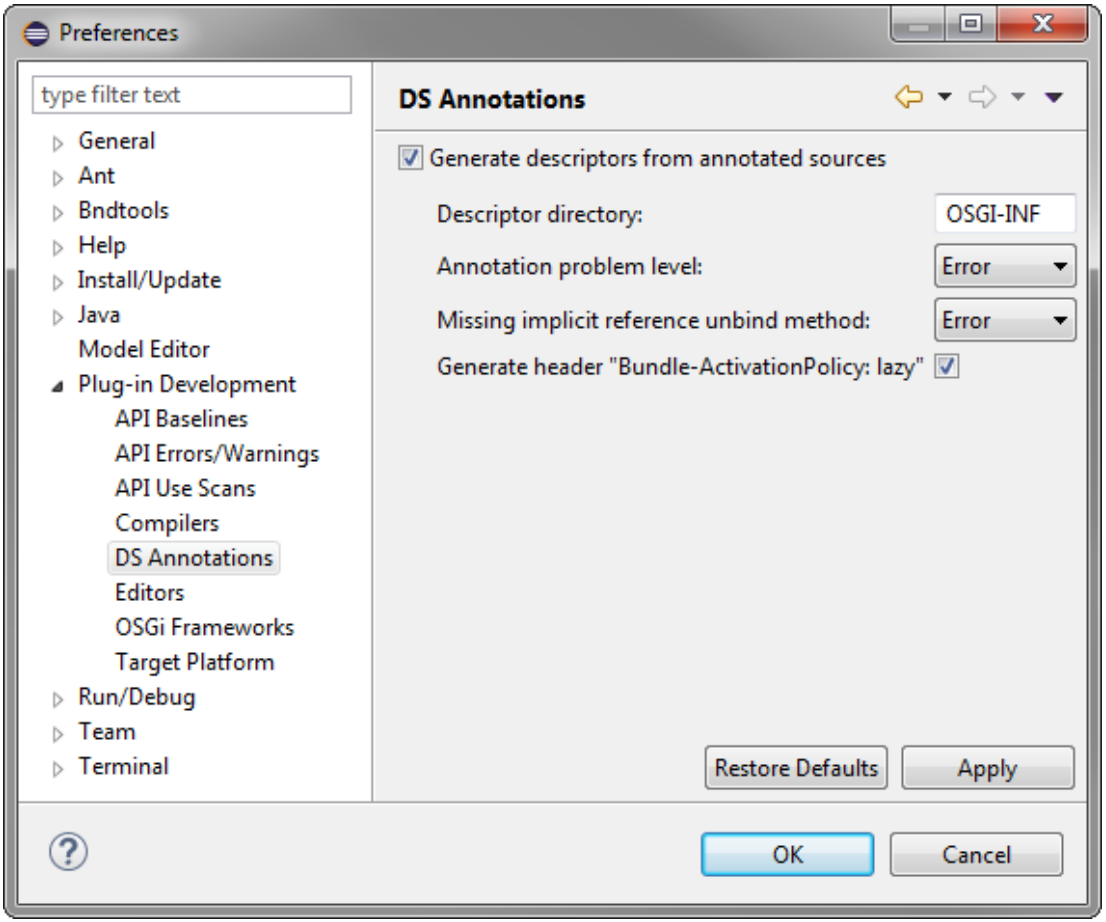
OSGi Declarative Services
Tooling

- **PDE**
  - Part of Eclipse (Neon)
  - Focus on MANIFEST.MF

- **Bndtools**
  - Additional Plug-in
  - http://bndtools.org/installation.html
  - Focus on bnd configurations
    → OSGi configurations / meta-data is generated

- ***Eclipse <= Mars***
  - *Declarative Services Annotations Support (Marketplace)*

OSGi Declarative Services
PDE – DS Annotations Support



BOSCH

IMPLEMENT & PUBLISH

BOSCH

OSGi Declarative Services
Service API

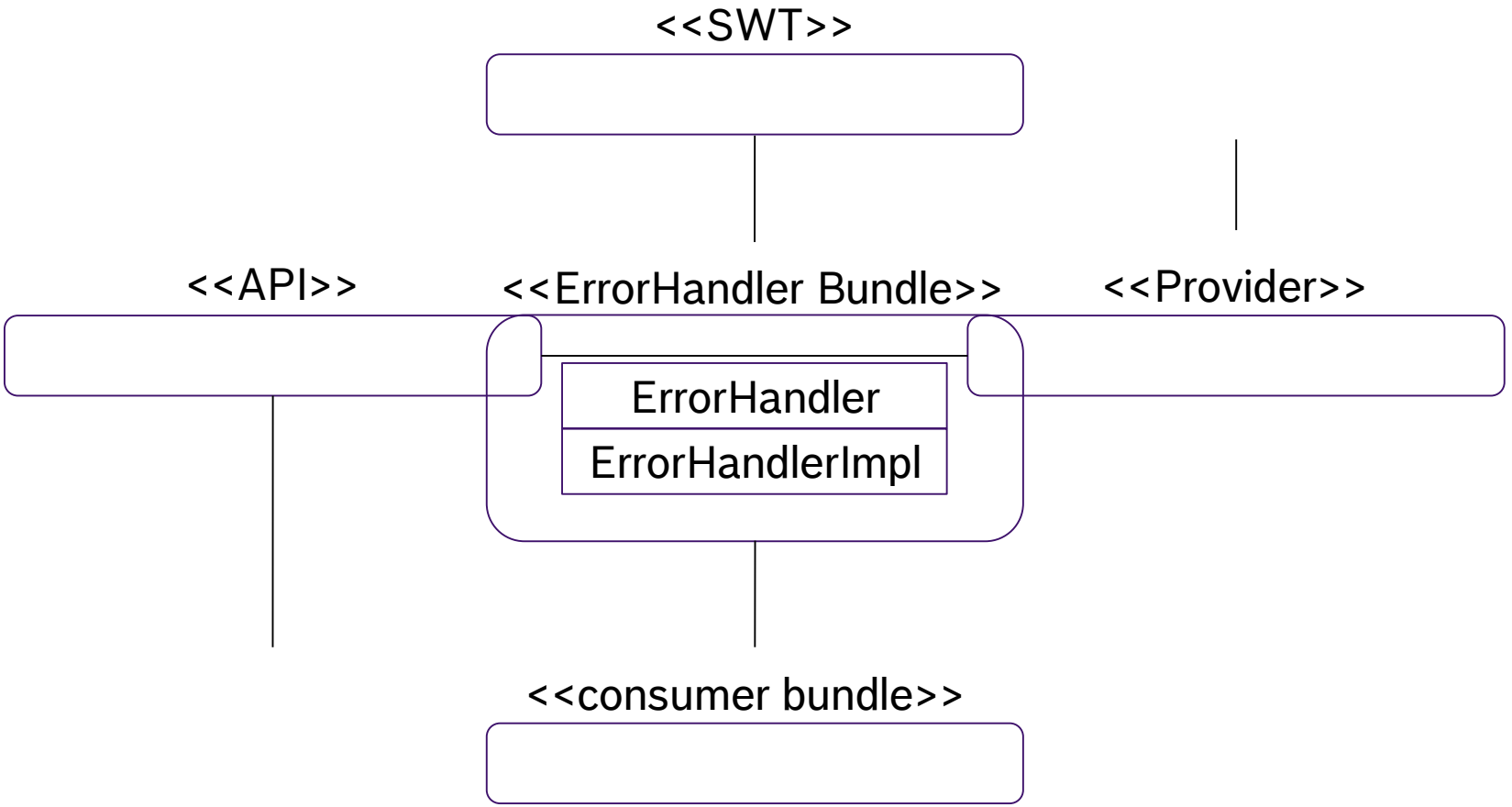
- Create a bundle for the service API (e.g. *org.fipro.inverter.api*)
- Create the service interface

```
public interface StringInverter {
    String invert(String input);
}
```

→ Make the service implementation exchangeable

→ Clean dependency hierarchy

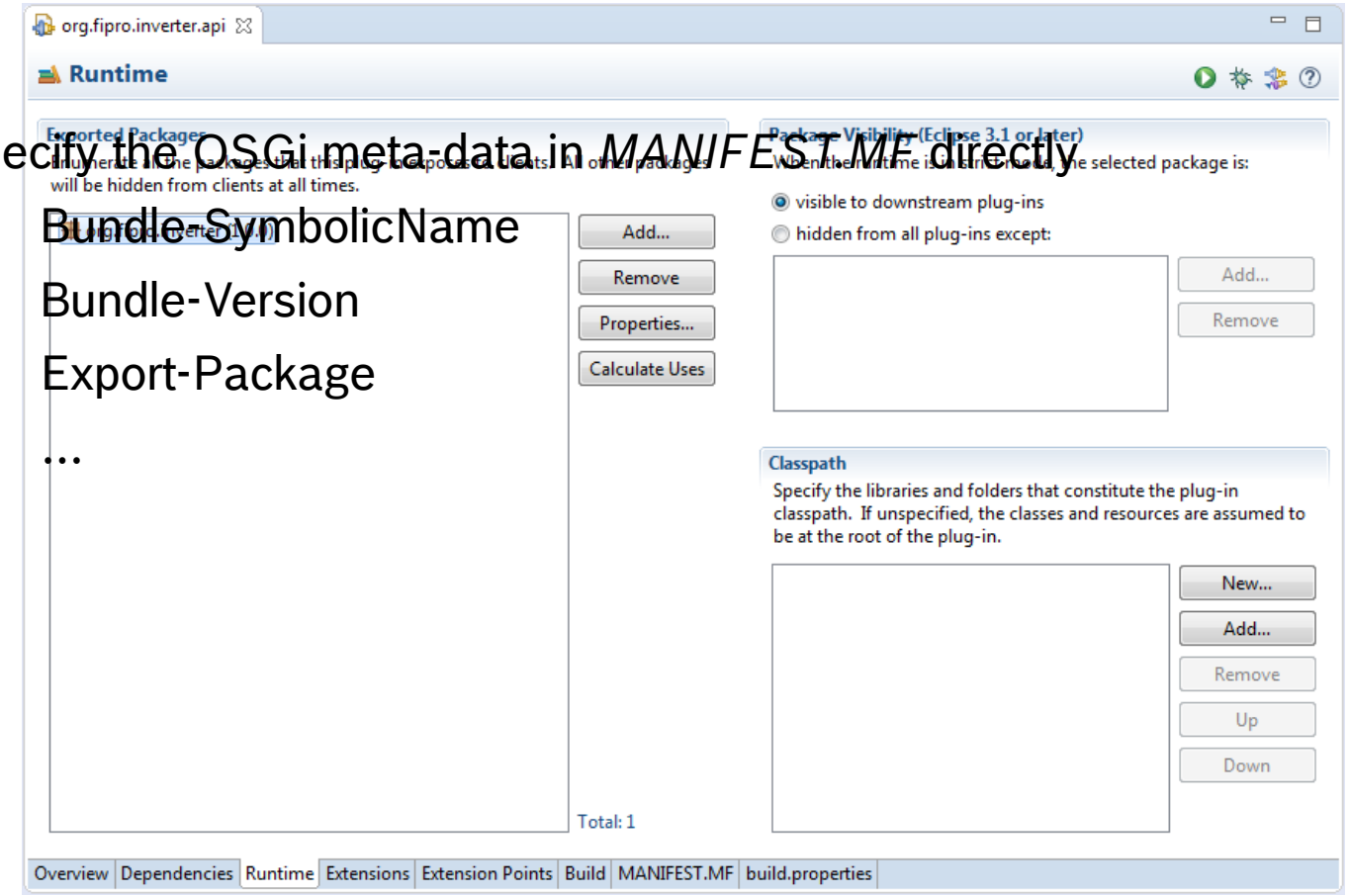
OSGi Declarative Services
Service API



OSGi Declarative Services
PDE vs. Bndtools

PDE

- Specify the OSGi meta-data in *MANIFEST.MF* directly
  - Bundle-SymbolicName
  - Bundle-Version
  - Export-Package
  - …

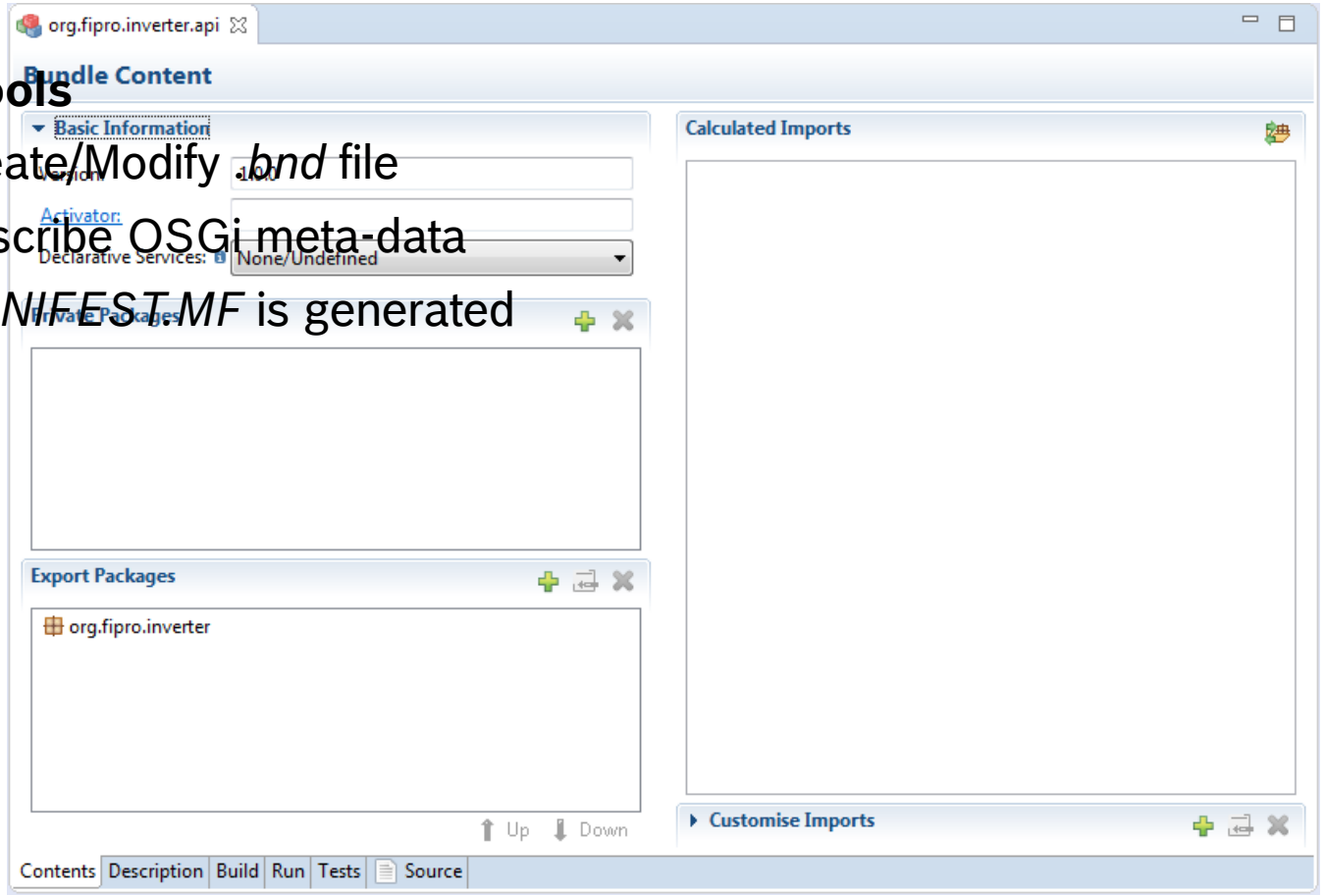


BOSCH

OSGi Declarative Services
PDE vs. Bndtools

Bndtools

- Create/Modify *.bnd* file
- Describe OSGi meta-data
- *MANIFEST.MF* is generated



BOSCH

OSGi Declarative Services
Service Provider

- Create a bundle for the service implementation (e.g. *org.fipro.inverter.provider*)
- Create the service implementation

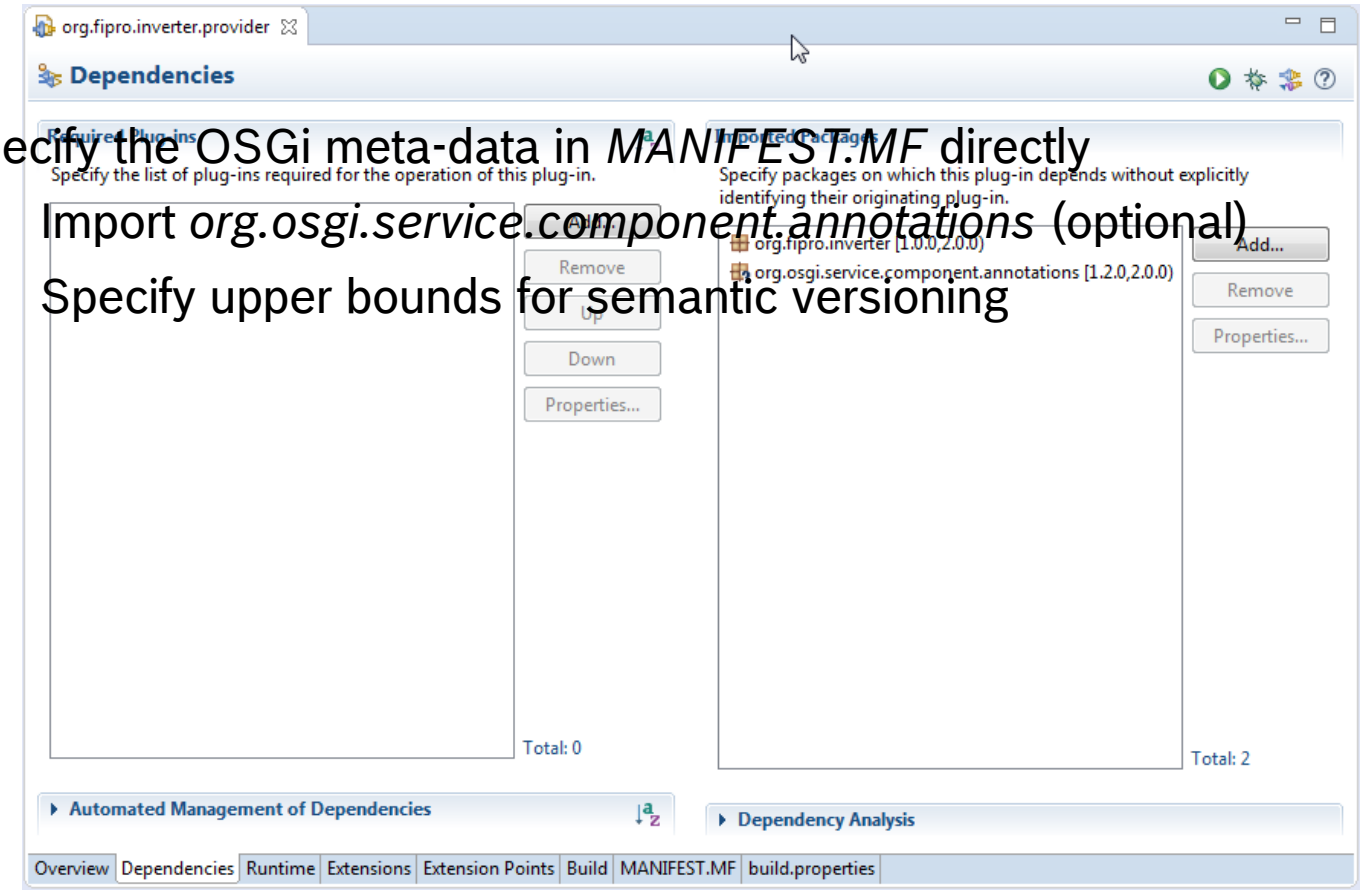
```
@Component
public class StringInverterImpl implements StringInverter {

    @Override
    public String invert(String input) {
            return new StringBuilder(input).reverse().toString();
    }
}
```

OSGi Declarative Services
PDE vs. Bndtools

PDE

- Specify the OSGi meta-data in *MANIFEST.MF* directly
  - Import *org.osgi.service.component.annotations* (optional)
  - Specify upper bounds for semantic versioning

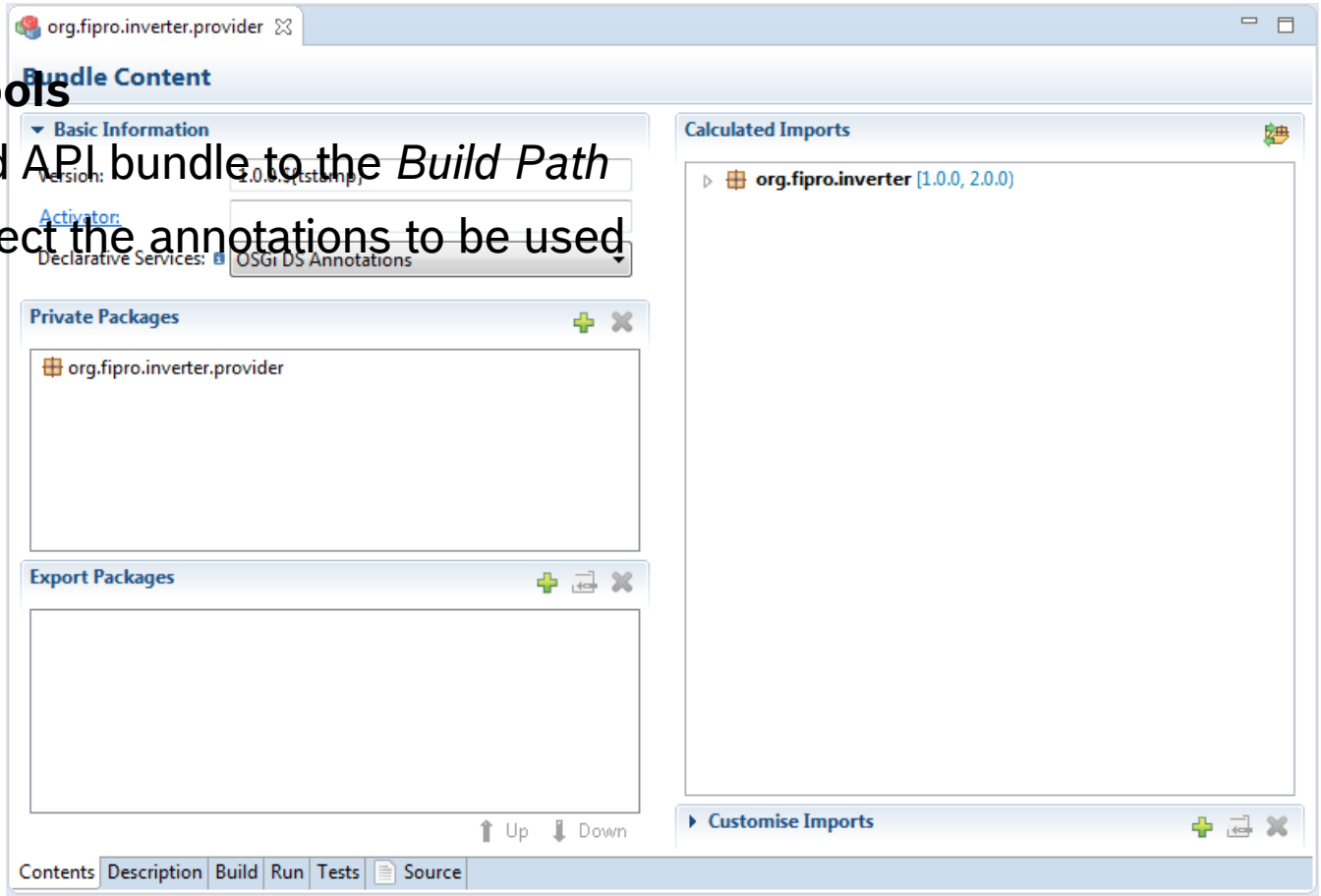


BOSCH

OSGi Declarative Services
PDE vs. Bndtools

Bndtools

- Add API bundle to the *Build Path*
- Select the annotations to be used



BOSCH

OSGi Declarative Services
@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*)

OSGi Declarative Services
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!

OSGi Declarative Services
@Component

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

OSGi Declarative Services
@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
}
```

OSGi Declarative Services
Service Consumer

- Create a bundle for the service consumer (e.g. *org.fipro.inverter.command*)
- Create the service consumer implementation

```
@Component(property= {"osgi.command.scope:String=fipro",
                      "osgi.command.function:String=invert"},
    service=StringInverterCommand.class
)
public class StringInverterCommand {

    private StringInverter inverter;

    @Reference
    void bindStringInverter(StringInverter inverter) { this.inverter = inverter; }

    public void invert(String input) { System.out.println(inverter.invert(input)); }
}
```

OSGi Declarative Services
@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*

OSGi Declarative Services
@Reference

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

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

   private StringInverter inverter;

   @Reference
   void bindStringInverter(StringInverter inverter) { this.inverter = inverter; }

   void updatedStringInverter(
            StringInverter inverter, Map<String, ?> properties) { //do something }

   void unbindStringInverter(StringInverter inverter) { this.inverter = null; }

   public void invert(String input) { System.out.println(inverter.invert(input)); }
}
```

OSGi Declarative Services
@Reference - Event Method Parameter

- `ServiceReference`
- <service type>
- <service type> + `Map<String, ?>`

With DS 1.3

- `ComponentServiceObjects`
- Different variations of the parameter list

OSGi Declarative Services
@Reference

- ***Field Strategy*** (*DS 1.3*) – using instance fields

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

    @Reference
    private StringInverter inverter;

    public void invert(String input) {
            System.out.println(inverter.invert(input));
    }
}
```

OSGi Declarative Services
@Reference

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

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

   private ComponentContext context;
   private ServiceReference<StringInverter> reference;

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

   @Reference
   void setStringInverter(ServiceReference<StringInverter> reference) { this.reference = reference; }

   public void invert(String input) {
            StringInverter inverter =
                        (StringInverter) context.locateService("StringInverter", reference);
            ...
   }
}
```

OSGi Declarative Services
@Reference

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

```
@Component(...
    reference=@Reference(name="inverter", service=StringInverter.class)
)
public class StringInverterCommand {

    private ComponentContext context;

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

    public void invert(String input) {
        StringInverter inverter = (StringInverter) context.locateService("inverter");
        ...
    }
}
```

OSGi Declarative Services
@Reference

Type Element	Default	Type Element	Default

cardinality	• 1:1 for event methods
	• 1:1 for non-collection fields, 0..n for collections
		unbind	`unbind<name>`
			`unset<name>`
			`remove<name>`

name	• bind event method name without bind prefix • name of the field	**updated**	`updated<name>` if such a method exists
policy	STATIC		
policyOption	RELUCTANT	**bind** (*since 1.3*)	the name of the annotated method or empty
service	full qualified class name of the referenced service	**field** (*since 1.3*)	the name of the annotated field or empty
target	empty String	**fieldOption** (*since 1.3*)	REPLACE
		scope (*since 1.3*)	BUNDLE

BOSCH

OSGi Declarative Services
Further information

- OSGi Compendium Specification
  https://www.osgi.org/developer/specifications/

- enRoute Documentation
  http://enroute.osgi.org/book/210-doc.html

- Blog posts
  http://blog.vogella.com/2016/06/21/getting-started-with-osgi-declarative-services/
  http://blog.vogella.com/2016/07/04/osgi-component-testing/