

# Practical OSGi

Sten Roger Sandvik  
Enonic AS

*srs@enonic.com*



# What is OSGi?

- **A module system for Java**  
The Dynamic Module System for Java™.
- **Dynamic modules (bundles)**  
Installing, starting, stopping, updating and uninstalling modules at runtime.
- **Service oriented**  
Services can be registered and consumed at runtime.
- **A worldwide standard**  
OSGi Alliance. Release R1 in 2000. Currenty at release R4.1.

# Key Benefits

- **No more “JAR” hell**  
Can have multiple versions of same library.
- **Reuse packaged modules**  
Allows 3rd party prepackaged modules to be installed.
- **Less server restarts**  
Update modules at runtime.
- **Simplifies multi-team projects**  
Can partition the application into smaller modules.

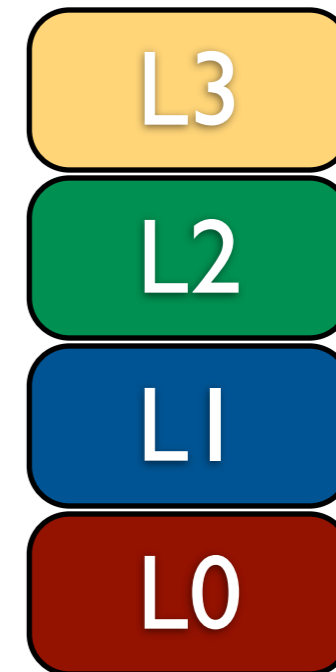
# Key Benefits (2)

- **Enables smaller systems**  
By breaking it into smaller pieces (some optional).
- **Multiple runtime options**  
Standalone, Client, Server, Embedded.
- **Multiple implementations**  
Eclipse Equinox, Apache Felix, Knopflerfish.
- **Very high adaption rate**  
Apache, Sun, Oracle, IBM, BMW and Enonic are all using it.

# Theory

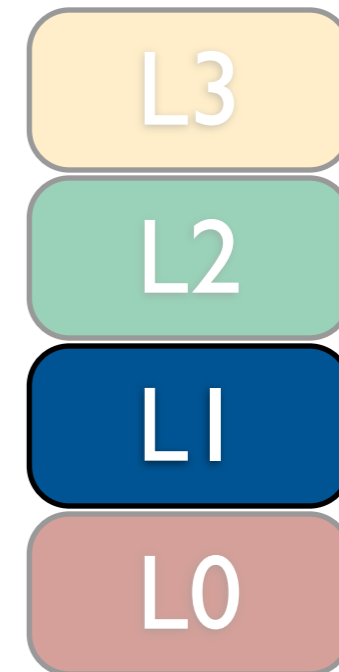
# Framework Layers

- **L3 - Service**  
Publish/find/bind service model to decouple bundles.
- **L2 - Lifecycle**  
Independent lifecycle of bundles without JVM restarts.
- **L1 - Module**  
A module (or bundle) uses classes from other bundles in a controlled way.
- **L0 - Execution environment**  
Well defined profiles (JavaSE, CDC) that define the JVM environment.



# Module Layer

- **Unit of deployment**  
A module is called a bundle in OSGi.
- **Standard JAR file with metadata**  
Metadata in META-INF/MANIFEST.MF
- **Seperate classloader per bundle**  
Class sharing at the Java package level.
- **Share only “interface” packages**  
Share only API between bundles.



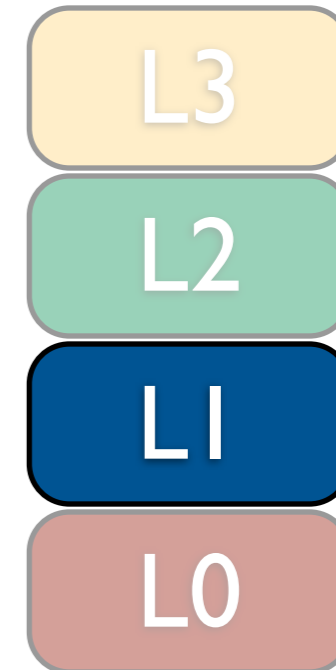
# Module Layer (2)

- **Multi-version support**

Different bundles can see different versions.

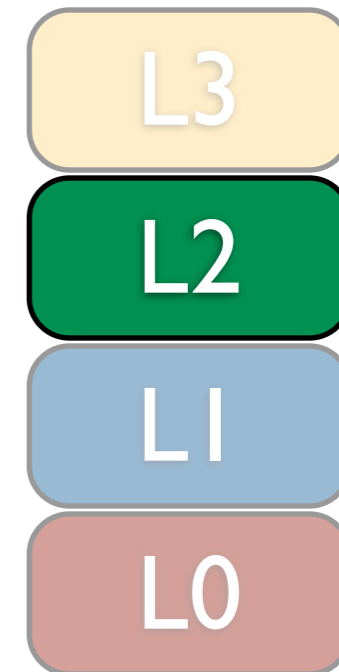
- **Manifest example**

```
Bundle-Name: Example
Bundle-SymbolicName: net.foo.common
Bundle-Version: 1.0.0
Import-Package:
  org.osgi.framework;version="1.3",
  net.foo.api;version="1.0.0"
Export-Package:
  net.foo.common;version="1.0.0"
Private-Package:
  net.foo.common.internal,
  net.foo.common.internal.codec
Bundle-ManifestVersion: 2
```



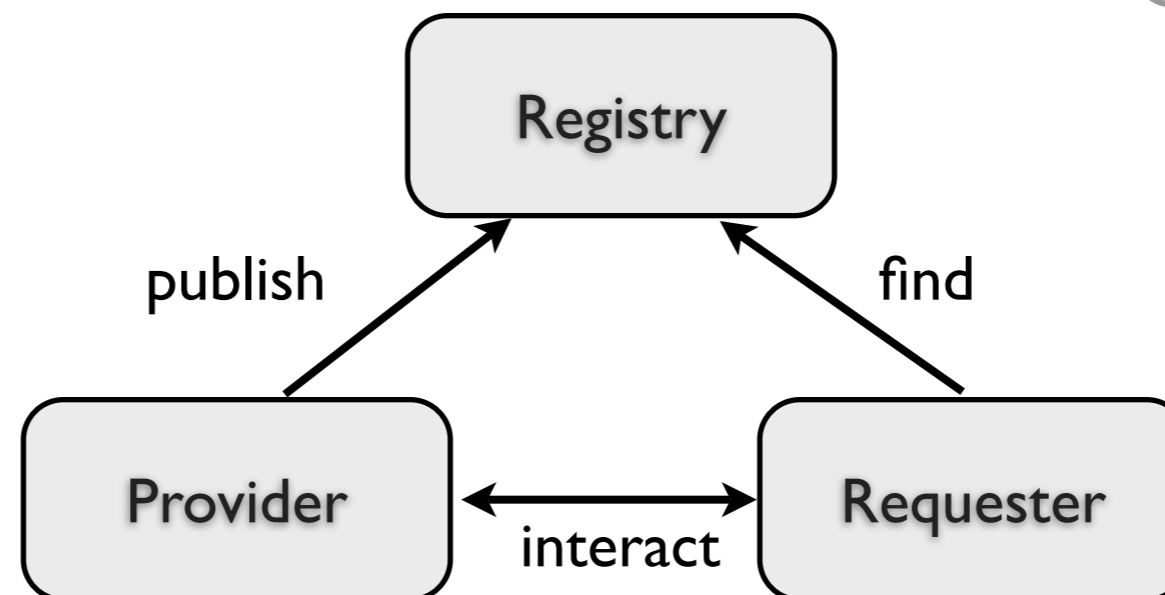
# Lifecycle Layer

- **Managed lifecycle**  
Managed states for each bundle.
- **Update existing bundles**  
Dynamically install, start, update and uninstall.



# Service Layer

- **Service registry**  
Publish/find/bind services at runtime.
- **Bind interfaces**  
Bind interfaces - not implementations.



# Practice

# Password Encoder

- **Password encoder application**  
Simple JVM client that uses password encoders.
- **Encoders are pluggable**  
Multiple implementations. Can be stopped/started at runtime.

# Project Structure

- **API Bundle**

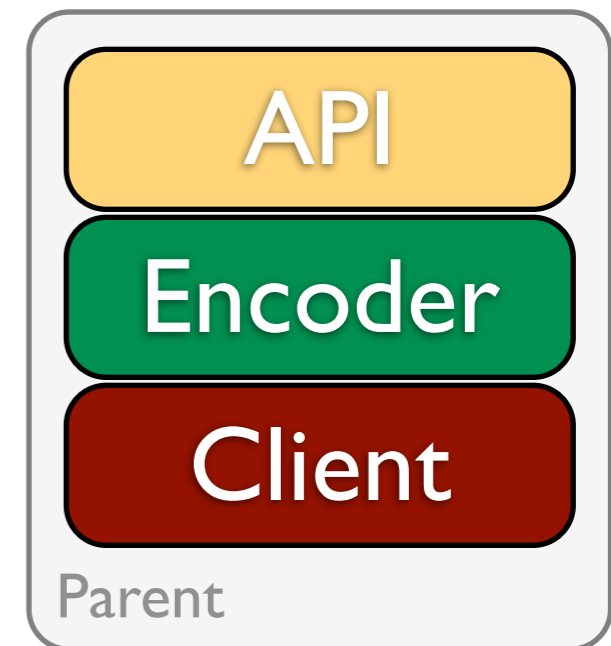
Holds all public available APIs.

- **Encoder Bundle**

Provides a password encoder implementation.

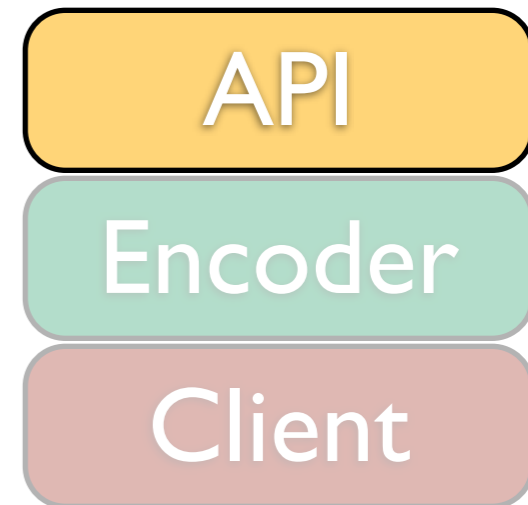
- **Client Bundle**

Client bundle that uses one or more password encoders.



# API Bundle

- **Provides the API**  
Simple PasswordEncoder interface.
- **Exports package**  
Export the API (*net.foo.api*) package so other bundles can see it.



# Password Encoder

```
package net.foo.api;

/**
 * Simple interface that defines the password
 * encoder.
 */
public interface PasswordEncoder
{
    /**
     * Returns the name of this encoder.
     */
    public String getName();

    /**
     * Encode the password and return the encoded password.
     */
    public String encode(String password);
}
```

API

Encoder

Client

# Manifest

- Exports the package

Exports package (*net.foo.api*) with right version.

```
...  
Bundle-Name: Example - API  
Bundle-SymbolicName: net.foo.api  
Export-Package:  
  net.foo.api;version="1.0.0"  
...
```

API

Encoder

Client

# Encoder Bundle

- **Uses the API**

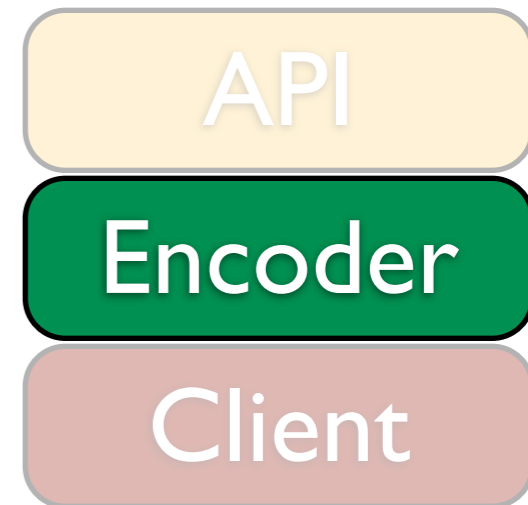
Imports proper API package (*net.foo.api*).

- **Registers a provider**

Registers a PasswordEncoder provider implementation as a service.

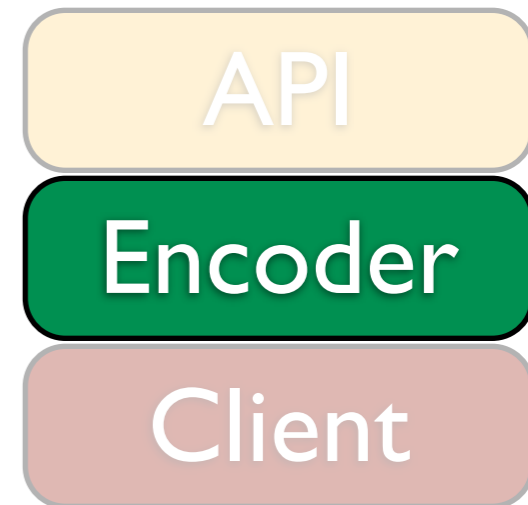
- **Hides the implementation**

Hides implementation from other bundles by using Private-Package.



# Register a Service

- **Use a BundleActivator**  
Implement bundle activator that handle lifecycle.
- **Register on start**  
Register service on when bundle is started.
- **Unregister at stop**  
Unregister the service when bundle is stopped.
- **Add proper metadata**  
Do not forget to add Bundle-Activator metadata in manifest.



# Bundle Activator

```
package net.foo.encoder;

import net.foo.api.*;
import org.osgi.framework.*;

public class Activator
    implements BundleActivator
{
    private ServiceRegistration reg;

    public void start(BundleContext context)
    {
        this.reg = context.registerService(
            PasswordEncoder.class.getName(),
            new ReversePasswordEncoder(), null);
    }

    public void stop(BundleContext context)
    {
        this.reg.unregister();
    }
}
```

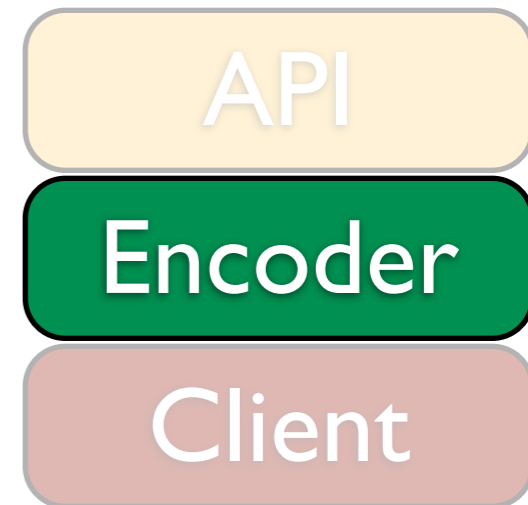
API

Encoder

Client

# Manifest

- **Declare BundleActivator**  
Set the bundle activator classname in manifest.
- **Import used packages**  
Import *net.foo.api* and *org.osgi.framework* packages with right version.
- **Hide implementation package**  
Declare *net.foo.encoder* package as a private package.



# Manifest (2)

```
...  
Bundle-Name: Example - Encoder  
Bundle-SymbolicName: net.foo.encoder  
Bundle-Activator:  
    net.foo.encoder.Activator  
Import-Package:  
    net.foo.api;version="1.0.0",  
    org.osgi.framework;version="1.3"  
Private-Package:  
    net.foo.encoder  
...
```

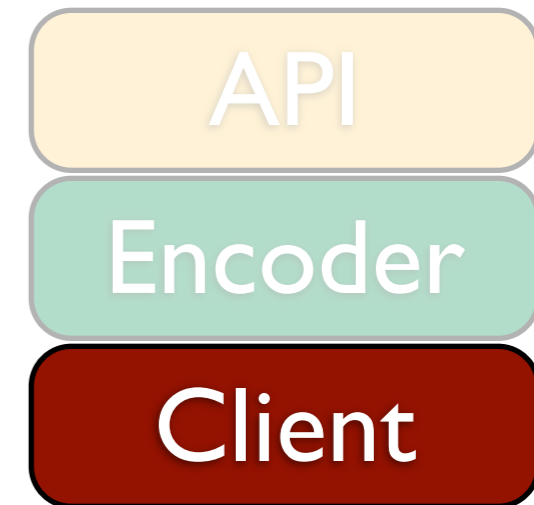
API

Encoder

Client

# Client Bundle

- **Uses the API**  
Does not see the encoder implementation.
- **Track PasswordEncoder's**  
Track available password encoder providers.
- **Encodes password based on encoder**  
Finds right encoder and encodes the password.



# Locating Services

- **Services are dynamic**

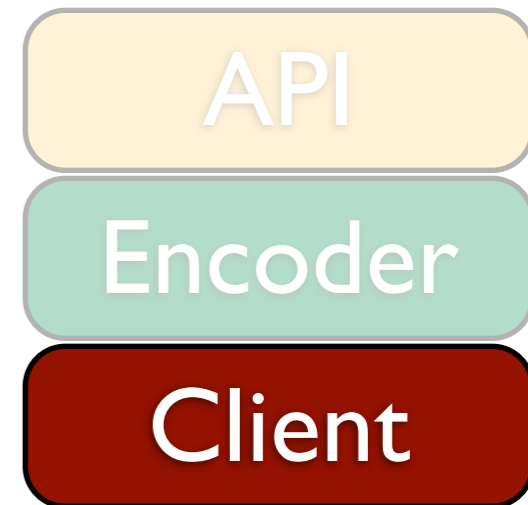
All services can be added and removed runtime.

- **Not always available on init**

Do not assume that you can always obtain a service during initialization.

- **Use ServiceTracker**

Use ServiceTracker to track available services. Remember to open() and close() the tracker.



# Using ServiceTracker

```
public class Activator
    implements BundleActivator
{
    private ServiceTracker tracker;

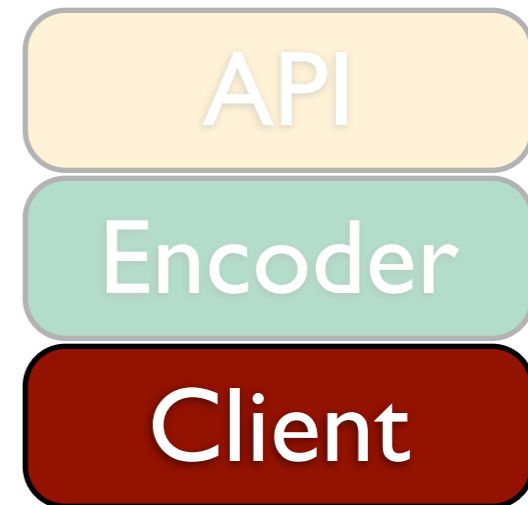
    public void start(BundleContext context)
    {
        tracker = new ServiceTracker(
            context, PasswordEncoder.class.getName(), null);
        tracker.open();
    }

    public void stop(BundleContext context)
    {
        tracker.close();
    }
}
```

```
public class MyConsumer
{
    private ServiceTracker tracker;

    ...

    public void consumeServices()
    {
        Object[] services = tracker.getServices();
    }
}
```



# Client Implementation

```
public class Client
{
    private ServiceTracker tracker;

    ...

    public String encode(String method, String password)
    {
        for (Object service : tracker.getServices()) {
            PasswordEncoder enc = (PasswordEncoder)service;

            if (enc.getName().equals(method)) {
                return enc.encode(password);
            }
        }

        throw new IllegalArgumentException("Provider not found");
    }
}
```

API

Encoder

Client

# Manifest

```
...  
Bundle-Name: Example - Client  
Bundle-SymbolicName: net.foo.client  
Bundle-Activator:  
    net.foo.client.Activator  
Import-Package:  
    net.foo.api;version="1.0.0",  
    org.osgi.framework;version="1.3",  
    org.osgi.util.tracker;version="1.3"  
Private-Package:  
    net.foo.client  
...
```

API

Encoder

Client

# Improvements

- **Use a declarative service model**  
Hides OSGi details. Makes service tracking easy. Spring DM is one implementation.
- **Use automatic manifest creation**  
Use Maven Bundle Plugin (Apache Felix Project) to create and validate manifest entries at build time.

# Demo

# Tools Used

- **Maven 2**  
Build the project using Maven 2 using standard layout.
- **Maven Bundle Plugin**  
Creates and validates the MANIFEST.MF. Based on BND.
- **Maven Pax Plugin**  
Running the OSGi application using Pax Runner.
- **Felix OSGi Container**  
Implementation of R4.1 specification.

# 3rd Party Bundles

- **Pax Web**

Web container based on Jetty. Implementation of HttpService.

- **Pax Logging**

Logging abstraction and LogService implementation. Possible to use SLF4J, Commons Logging and Log4J in bundles.

- **Felix WebConsole**

Web Console to visualize bundles and services.

Q?

*srs@enonic.com*