

Extracted from:

Modular Java

Creating Flexible Applications with OSGi and Spring

This PDF file contains pages extracted from Modular Java, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2009 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The
Pragmatic
Programmers

Modular Java

Creating Flexible Applications
with OSGi and Spring



Craig Walls

Edited by Jacquelyn Carter



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

<http://www.pragprog.com>

Copyright © 2009 Craig Walls.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-10: 1-934356-40-9

ISBN-13: 978-1934356-40-1

Printed on acid-free paper.

P1.0 printing, May 2009

Version: 2009-6-2

To get started with Spring-DM, we'll need to add these bundles to our project:

```
dwmjs% pax-import-bundle -g org.springframework.osgi -a spring-osgi-extender \
?                               -v 1.2.0 -- -DimportTransitive -DwidenScope
[INFO] Scanning for projects...
...
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 8 seconds
[INFO] Finished at: Fri Mar 20 15:33:34 CDT 2009
[INFO] Final Memory: 9M/18M
[INFO] -----
dwmjs%
```

Here we've asked Pax Construct to add version 1.2.0 of the Spring-DM extender bundle (identified with a group ID of `org.springframework.osgi` and an artifact ID of `org.springframework.osgi.extender`) to the project. In addition to the Spring-DM extender bundle itself, we've also asked that `pax-import-bundle` also pull in transitive dependencies (`-DimportTransitive`) and to consider all compile and runtime dependencies as potential bundles (`-DwidenScope`).

The Spring-DM bundles are now in place and are ready to help us declaratively publish the index service.

6.2 Declaring Services

The first step in declaring a service in Spring-DM is to wire a bean in the Spring application context. In Spring, a bean is any object (not necessarily a JavaBean) that is instantiated and managed by the Spring Framework. A common way of configuring the beans that Spring creates is to define a Spring application context in an XML file. For example, consider this Spring configuration XML (`index-context.xml`) that we'll use to define an application context for the index service bundle:

[Download](#) `dwmjs/index/src/main/resources/META-INF/spring/index-context.xml`

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:compass="http://www.compass-project.org/schema/spring-core-config"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.compass-project.org/schema/spring-core-config
    http://www.compass-project.org/schema/spring-compass-core-config-2.0.xsd">
  <bean id="indexService"
    class="dwmj.index.internal.IndexServiceImpl">
    <constructor-arg ref="compass" />
  </bean>
```

```

<compass:compass name="compass" >
  <compass:connection>
    <compass:file path="/tmp/dudeindex" />
  </compass:connection>
  <compass:mappings>
    <compass:class name="dwmj.domain.JarFile"/>
  </compass:mappings>
</compass:compass>

  <compass:session id="compassSession" />
</beans>

```

Here we've declared two beans. The first is defined with the `<bean>` element. This bean tells Spring to create an instance of `IndexServiceImpl` and to give it an ID of `indexService`. What's especially interesting about this bean is that we're telling Spring to instantiate it by calling a single-argument constructor and passing in a reference to another bean. Specifically, Spring should construct `IndexServiceImpl` with a reference to a bean whose ID is `compass`.

That brings us to the second bean. Instead of using a generic `<bean>` element to declare the `compass` bean, we're using elements from a Compass-specific configuration namespace provided as part of the Compass library. Ultimately, this declaration creates a bean of type `org.compass.core.Compass`, suitable for the first argument of the `IndexServiceImpl` constructor.

As mentioned before, Spring-DM creates an application context by reading all XML files in the `META-INF/spring` directory. Since we're building the bundle using Maven, we'll need to place `index-context.xml` in the `src/main/resources/META-INF/spring` directory of the index bundle project.

But it won't be alone. In addition to the core Spring configuration file, we'll also create a separate Spring configuration file (`index-osgi.xml`) that tells Spring-DM to publish the `indexService` bean to the OSGi service registry:

[Download](#) `dwmjs/index/src/main/resources/META-INF/spring/index-osgi.xml`

```

<beans:beans xmlns="http://www.springframework.org/schema/osgi"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/osgi
    http://www.springframework.org/schema/osgi/spring-osgi.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <service ref="indexService"
    interface="dwmj.index.IndexService" />

</beans:beans>

```

Using Spring-DM with Java 1.4

In Spring-DM, it's common for beans from different application contexts to interact with each other concurrently. To avoid deadlocks when beans are requested from the application contexts, Spring-DM needs concurrent collections. Java 1.5 and later provide concurrent collections out of the box. But Java 1.4 does not.

To add concurrent collection classes for Java 1.4, you'll need to add the Backport bundle. First, because the Backport libraries in the central Maven repository aren't OSGi-ready bundles, you'll need to add the Spring-DM repository:

```
dwmjs% pax-add-repository -i spring-osgi -u \
?      http://s3.amazonaws.com/maven.springframework.org/osgi \
?      -- -Dsnapshots
```

Then import the Backport bundle into the project:

```
dwmjs% pax-import-bundle -g org.springframework.osgi -a \
?      backport-util-concurrent.osgi -v 3.0-SNAPSHOT -- \
?      "-DimportPackage=sun.misc;resolution:=optional,*"
...
dwmjs%
```

To keep the OSGi-specific configuration separate from the generic bean definitions, I've placed this service declaration in a separate configuration file. The `<service>` element declares that the bean referenced by the `ref=` attribute should be published to the OSGi service registry with the interface specified in the `interface=` attribute. In this case, it's the index service bean that we declared in `index-context.xml`, which should be published with the `dwmj.index.IndexService` interface.

And that simple bit of Spring configuration is all we need to do to declare the index service bean as an OSGi service. You've no doubt noticed that this is simpler than programmatically publishing it using a bundle activator. All of the hassles of working directly with the OSGi API go away and are replaced with a simple entry in a Spring application context configuration file.

Speaking of not having to deal with the OSGi API, we no longer need the index bundle's activator. We needed it only to create and publish the index service.

But since Spring-DM's handling that for us now, we can get rid of it:

```
dwmjs% cd index
index% rm src/main/java/dwmj/index/internal/IndexServiceActivator.java
index%
```

We'll also need to delete the Bundle-Activator: entry from the osgi.bnd file.

Now that we've swapped out the bundle activator for a Spring-DM service declaration, let's rebuild the index service...

```
index% mvn install
[INFO] Scanning for projects...
...
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 7 seconds
[INFO] Finished at: Fri Mar 20 15:47:34 CDT 2009
[INFO] Final Memory: 13M/31M
[INFO] -----
index%
```

... and then provision it:

```
index% cd ..
dwmjs% pax-provision
[INFO] Scanning for projects...
...
Caused by: java.lang.ClassNotFoundException:
    org.compass.core.config.binding.metadata.AsmMetadataReader
        not found from bundle [com.dudewheresmyjar.index]
...
osgi>
```

Oops! It looks like our index bundle had a little trouble getting started. Now that we're using Compass' configuration namespace for Spring, our bundle needs to import some packages that we're not currently importing. But wait—the index service is already using Compass in some capacity, and we haven't had to import any Compass packages before. Why must we import Compass packages now?

The answer is a bit nonobvious. As you'll recall, our build is using the BND tool to generate the MANIFEST.MF file from the osgi.bnd file. When we were programmatically working with Compass in the bundle activator, BND was able to figure out what packages to import by analyzing the activator and the service classes. But now the activator class has gone away, and we're declaring much of the Compass stuff in the Spring configuration file.

Unfortunately, BND doesn't analyze the Spring configuration file when putting together its list of packages to import. So, we'll have to add those imports to `osgi.bnd` ourselves:

[Download](#) `dwmjs/index/osgi.bnd`

```
Import-Package: *, \
    org.compass.core.engine.naming, \
    org.compass.core.executor.concurrent, \
    org.compass.core.cache.first, \
    org.compass.core.lucene.engine.analyzer, \
    org.compass.core.lucene.engine.optimizer, \
    org.compass.core.transaction, \
    org.apache.lucene.index, \
    org.apache.lucene, \
    org.apache.lucene.document, \
    org.apache.lucene.queryParser, \
    org.apache.lucene.search, \
    org.apache.lucene.store, \
    org.apache.lucene.util, \
    org.compass.core.config.binding.metadata, \
    org.compass.core.json.impl.converter
```

The first item in the import list is `*`, which tells BND to import all packages that it finds while analyzing Java classes—the default import behavior. What follows are the packages that are needed by Compass.¹ Let's build the index bundle and try provisioning it again:

```
dwmjs% pax-provision
[INFO] Scanning for projects...
...
Caused by: java.lang.NoClassDefFoundError:
    org/springframework/transaction/PlatformTransactionManager
...
osgi>
```

We have one more hurdle to overcome. It seems that Spring can't create the `compass` bean because it can't find `org.springframework.transaction.PlatformTransactionManager`. What? Spring cannot find one of its own classes?

As it turns out, `PlatformTransactionManager` resides in a separate bundle from the Spring bundles that we've already installed. To get past this problem, we're going to need to add Spring's transaction support bundle to our project.

1. I figured out what packages are needed by a tedious trial and error effort. I'm sparing you the effort of walking you through that exercise. But if you'd like to try it yourself, you can start by importing `org.compass.core.engine.naming`—the package containing the class that was the subject of the `ClassNotFoundException` we encountered—and following the breadcrumbs from there.


```

dwmjs% pax-import-bundle -g org.springframework -a spring-tx -v 2.5.6
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building com.dudewheresmyjar.dwmj (OSGi project)
[INFO]   task-segment: [org.ops4j:maven-pax-plugin:1.4:import-bundle]
[INFO]   (aggregator-style)
[INFO] -----
[INFO] [pax:import-bundle]
[INFO] Importing Spring Framework: Transaction to
[INFO]   com.dudewheresmyjar.dwmj.build:provision:pom:1.0.0-SNAPSHOT
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 5 seconds
[INFO] Finished at: Fri Mar 20 15:54:07 CDT 2009
[INFO] Final Memory: 8M/18M
[INFO] -----
dwmjs%

```

With the Spring transaction support bundle in place, let's try to provision all of our bundles one more time:

```

dwmjs% pax-provision
[INFO] Scanning for projects...
...
osgi> ss

```

Framework is launched.

id	State	Bundle
0	ACTIVE	org.eclipse.osgi_3.4.2.R34x_v20080826-1230
1	ACTIVE	org.eclipse.osgi.util_3.1.300.v20080303
2	ACTIVE	org.eclipse.osgi.services_3.1.200.v20070605
3	ACTIVE	org.ops4j.pax.logging.pax-logging-api_1.3.0
4	ACTIVE	org.ops4j.pax.logging.pax-logging-service_1.3.0
5	ACTIVE	org.springframework.osgi.extender_1.2.0
6	ACTIVE	org.springframework.osgi.core_1.2.0
7	ACTIVE	org.springframework.osgi.io_1.2.0
8	ACTIVE	com.springsource.slf4j.org.apache.commons.logging_1.5.0
9	ACTIVE	com.springsource.slf4j.api_1.5.0 Fragments=10
10	RESOLVED	com.springsource.slf4j.log4j_1.5.0 Master=9
11	ACTIVE	org.springframework.aop_2.5.6
12	ACTIVE	org.springframework.beans_2.5.6
13	ACTIVE	org.springframework.context_2.5.6
14	ACTIVE	org.springframework.core_2.5.6
15	ACTIVE	org.springframework.test_2.5.6
16	ACTIVE	com.springsource.org.aopalliance_1.0.0
17	ACTIVE	org.springframework.transaction_2.5.6
18	ACTIVE	com.dudewheresmyjar.domain_1.0.0.SNAPSHOT

```

19    ACTIVE    org.compass-project.compass_2.1.1
20    ACTIVE    com.dudewheresmyjar.index_1.0.0.SNAPSHOT
21    ACTIVE    com.dudewheresmyjar.spider_1.0.0.SNAPSHOT

```

osgi>

So far so good. There were no exceptions thrown that time, and all of our bundles are active. Let's use the bundle command to dig a little deeper into the index bundle to see whether it is publishing the index service:

```

osgi> bundle 20
initial@reference:file:com.dudewheresmyjar.index_1.0.0.SNAPSHOT.jar/ [20]
  Id=20, Status=ACTIVE      Data Root=/Users/wallsc/Projects/projects/dwmjs/
                           runner/equinox/org.eclipse.osgi/bundles/20/data

Registered Services
▶ {dwmj.index.IndexService}={org.springframework.osgi.bean.name=indexService,
▶   Bundle-SymbolicName=com.dudewheresmyjar.index,
▶   Bundle-Version=1.0.0.SNAPSHOT, service.id=26}
{org.springframework.osgi.context.DelegatedExecutionOsgiBundleApplicationContext,
org.springframework.osgi.context.ConfigurableOsgiBundleApplicationContext,
org.springframework.context.ConfigurableApplicationContext,
org.springframework.context.ApplicationContext,
org.springframework.context.Lifecycle,
org.springframework.beans.factory.ListableBeanFactory,
org.springframework.beans.factory.HierarchicalBeanFactory,
org.springframework.context.MessageSource,
org.springframework.context.ApplicationEventPublisher,
org.springframework.core.io.support.ResourcePatternResolver,
org.springframework.beans.factory.BeanFactory,
org.springframework.core.io.ResourceLoader,
org.springframework.beans.factory.DisposableBean}=
{org.springframework.context.service.name=com.dudewheresmyjar.index,
  Bundle-SymbolicName=com.dudewheresmyjar.index,
  Bundle-Version=1.0.0.SNAPSHOT, service.id=27}
...

```

osgi>

It looks like that worked, as evidenced by the first entry under the *Registered Services* header. Notice that there's a lot of information about the service, including the interface that it's published under, the bundle that publishes the service, and the Spring bean that provides the service.

You may have noticed that there's another entry under *Registered Services*—where'd that come from? In addition to publishing the services declared using the `<service>` element, Spring-DM also publishes the Spring application context as a service. And, it's published under a

How to Not Publish the Spring Context as a Service

If you'd rather not have a bundle's Spring context published as a service, you'll need to say so with the Spring-Context: header:

```
Spring-Context: META-INF/spring/*.xml;publish-context:=false
```

By setting the publish-context directive to false, we're asking Spring-DM to go ahead and load the Spring context using XML files in META-INF/spring, but not to publish the context in the OSGi service registry.

baker's dozen of interfaces, any of which you can use to retrieve the bundle's Spring context.

Now that we've converted the index bundle to use Spring-DM, let's turn our attention to the spider bundle to see whether Spring-DM can help us eliminate all of the code that we wrote to consume the index service.

6.3 Injecting Services into Consumers

As you'll recall, there's much more to consuming a service than publishing it. A service consumer must carefully deal with the transitivity of services to make sure that it's not trying to use a service that has gone away or that has been replaced with a newer version. All of that service management resulted in a lot of code in both the spider bundle's activator and in the spider implementation class.

Spring-DM was able to eliminate OSGi-specific code in our index bundle. Can it do the same for the spider bundle? You bet! In fact, as you'll soon see, consuming a service with Spring-DM isn't much different from publishing a service.

First things first. . . just as with the index bundle, we're no longer going to need the bundle activator for the spider bundle. So, let's go ahead and ditch it:

```
dwmjs% cd spider
spider% rm src/main/java/dwmj/spider/impl/SpiderActivator.java
```

Be sure to remove the Bundle-Activator: entry from osgi.bnd, too.

Now that the spider's bundle activator is gone, we no longer have a way to give the MavenSpider a service tracker to look up the index service.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Modular Java's Home Page

<http://pragprog.com/titles/cwosg>

Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates

<http://pragprog.com/updates>

Be notified when updates and new books become available.

Join the Community

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy

<http://pragprog.com/news>

Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/cwosg.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)