CAMEL DEPLOYMENT MODES

Most popular and useful modes are:

1. Maven goal (useful for testing purposes),
2. Main method (useful for JSE applications),
3. Web container (useful for web applications),
4. OSGi (useful for modular development).

Before I commence, I need sample applications to deploy them in different ways. Fortunately, Camel is distributed with a few archetypes for maven users. I'll use three of them:

- *camel-archetype-spring*,
- *camel-archetype-web*,
- *camel-archetype-blueprint*.

To generate the project, use the following command:

```
 mvn archetype:generate \
  -DarchetypeGroupId=org.apache.camel.archetypes \
  -DarchetypeArtifactId=*archetype-name* \
  -DarchetypeVersion=2.9.0 \
  -DarchetypeRepository=https://repository.apache.org/content/groups/snapshot
s-g
```

where **archetype-name** is the name of one of the above mentioned archetypes.

Maven goal

This is extremely simple: just create *camel-archetype-spring* and execute command *mvn camel:run*. That's it! We have a fully working Apache Camel application. Take a look at camel-context.xml.

Main method

If you need to create simple integration platform that will run as a classical JSE Main application, you need to use *Main* class from *org.apache.camel.main package*. Below there is a code for a very simple application that will print on your console *"Invoked at " + new Date()* every 2 seconds (2000 miliseconds).

```
import java.util.Date;
 import org.apache.camel.Exchange;
 import org.apache.camel.Processor;
 import org.apache.camel.builder.RouteBuilder;
 import org.apache.camel.main.Main;
 public class MainExample {
   private Main main;
   public static void main(String[] args) throws Exception {
     MainExample example = new MainExample();
     example.boot();
   }
   public void boot() throws Exception {
     main = new Main();
       ①
     main.enableHangupSupport();
       ②
     main.addRouteBuilder(new MyRouteBuilder());
       ③
     main.run();
   }
   private static class MyRouteBuilder extends RouteBuilder {
     @Override
     public void configure() throws Exception {
       from("timer:foo?delay=2000")
         .process(new Processor() {
           public void process(Exchange exchange) throws Exception {
             System.out.println("Invoked at " + new Date());
           }
         });
     }
   }
}
```

Line ① enables hangup support which means that you can terminate the JVM using ctrl + C.
Line ② adds route builder with simple logic.
Line ③ runs application.

Web container

Firstly create *camel-archetype-web* and then execute command *jetty:run*. The message *"Hello Web Application, how are you?"* will then appear every 10 seconds. Let's take a closer look at the Camel configuration and web descriptor.

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com
/xml/ns/j2ee/web-app_2_4.xsd">
   <display-name>Camel Routes</display-name>
   <!-- location of spring xml files -->
   <context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/applicationContext.xml</param-value>
   </context-param>
   <!-- the listener that kick-starts Spring -->
   <listener>
     <listener-class>org.springframework.web.context.ContextLoaderListener</l
istener-class>
   </listener>
 </web-app>
```

Web descriptor registers Spring context loader listener with specified context configuration location which contains Camel context. Camel context handler then starts all defined routes.

OSGi

OSGi is the specification that describe module and service based platform for the Java. For more information about OSGi, visit http://www.osgi.org/Specifications/HomePage .

To begin, you need to generate project from camel-archetype-blueprint. Single module (in our example jar) in OSGi is called bundle. Firstly let’s take a look at maven configuration file (pom.xml):

```
<plugin>
 <groupId>org.apache.felix</groupId>
 <artifactId>maven-bundle-plugin</artifactId>
 <version>2.3.4</version>
 <extensions>true</extensions>
 <configuration>
  <instructions>
   <Bundle-SymbolicName>camel-blueprint</Bundle-SymbolicName>
   <Private-Package>com.blogspot.mw.camel.blueprint.*</Private-Package>
   <Import-Package>*,org.apache.camel.osgi</Import-Package>
  </instructions>
 </configuration>
</plugin>
```

The code shows Apache Felix plugin execution configuration that will generate *MANIFEST.MF* file in *META-INF* directory. Private-Package indicates which of the available packages to copy into the bundle but not export. Import-Package indicates which of the available external packages to use. In our case we need to import *org.apache.camel.osgi*. Below you can see generated *MANIFEST.MF*:

```
Manifest-Version: 1.0
Export-Package: com.blogspot.mw.camel.blueprint
Bundle-Version: 1.0.0.SNAPSHOT
Build-Jdk: 1.6.0_25
Built-By: Michal
Tool: Bnd-1.15.0
Bnd-LastModified: 1338148449092
Bundle-Name: A Camel Blueprint Route
Bundle-ManifestVersion: 2
Created-By: Apache Maven Bundle Plugin
Import-Package: org.apache.camel.osgi,org.osgi.service.blueprint;versi
```

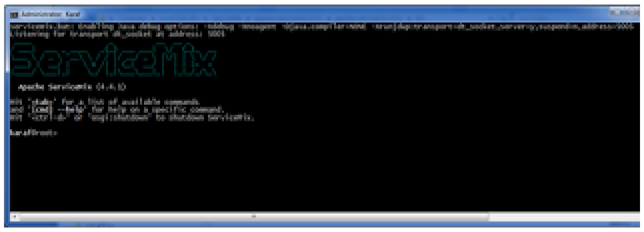
```
  on="[1.0.0,2.0.0)"

 Bundle-SymbolicName: camel-blueprint
```

The second file worth discussing is blueprint.xml from *src/main/resources/OSGI-INF/blueprint/blueprint.xml*. Blueprint specification describe information about dependency injection and inversion of control for OSGi environment.
As you can see in the code hereunder this is very similar to Spring configuration (in fact blueprint derives Spring configuration):

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"

     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

     xmlns:camel="http://camel.apache.org/schema/blueprint"

     xsi:schemaLocation="

     http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blu
eprint/v1.0.0/blueprint.xsd

     http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/
blueprint/camel-blueprint.xsd">

  <bean id="helloBean" class="com.blogspot.mw.camel.blueprint.HelloBean">

    <property name="say" value="Hi from Camel"/>

  </bean>

  <camelContext trace="false" id="blueprintContext" xmlns="http://camel.apach
e.org/schema/blueprint">

   <route id="timerToLog">

     <from uri="timer:foo?period=5000"/>

     <setBody>

       <method method="hello" ref="helloBean"></method>

     </setBody>

     <log message="The message contains ${body}"/>

     <to uri="mock:result"/>

   </route>

 </camelContext>

 </blueprint>
```

To run this example we have to download some OSGi runtime environment. I’ll use for this purpose Apache ServiceMix (http://servicemix.apache.org/).

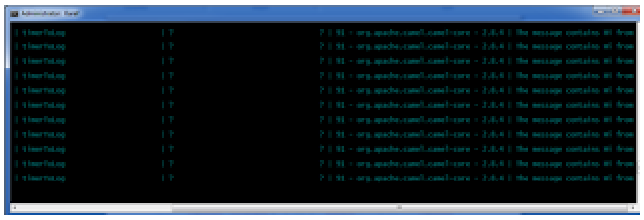
To run ServiceMix, execute servicemix.bat (or servicemix.sh if your operating system is unix based).



```
servicemix.bat: Enabling Java debug options: -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005
Listening for transport dt_socket at address: 5005

  Apache ServiceMix (4.4.1)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown ServiceMix.

karaf@root>
```

Next you have to install bundle into ServixeMix from Maven repository. In order to do this execute command *osgi:install -s mvn:project-group-id/project-articafact-id*. In my case that is *osgi:install –s mvn:com.blogspot.mw/camel-blueprint*.

Now you can display logs by typing command log:display.



```
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
| timerToLog                | ?                          ? | 91 - org.apache.camel.camel-core - 2.8.4 | The message contains Hi from
```

In the screen above you can see the logs so the application is up and running.