## Running Batch jobs in J2SE applications

<http://www.mastertheboss.com/javaee/batch-api/running-batch-jobs-in-j2se-applications>

This tutorial shows how you can run the **Java Batch API (JSR 352)** as part of a J2SE application.

The **Java Batch API (JSR 352)** allows executing Batch activities based on a **Job Specification Language (JSL)** using two main Programming models: Chunk Steps or Batchlets. I've already blogged some examples of [Chunk steps](http://www.mastertheboss.com/batch-api/batch-applications-tutorial-on-wildfly" \t "_blank)  and [BatchLets](http://www.mastertheboss.com/batch-api/batch-applications-tutorial-on-wildfly" \t "_blank) which are designed to run on WildFly application server.  
The implementation of Java Batch API (JSR 352) is provided by a project named **JBeret** which allows also executing Batch activities as part of Java standard edition applications. In this tutorial we will see a basic example of how to run a Batchlet from within a J2SE application.

# Defining the Batch Job

The first step is defining the job via the Job Specification Language (JSL). Let's create this file named **simplebatchlet.xml** in the folder **src\main\resources\META-INF\batch-jobs** of a Maven project:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <job id="simplebatchlet" xmlns="http://xmlns.jcp.org/xml/ns/javaee"      version="1.0">        <step id="step1">          <properties>              <property name="file" value="/home/jboss/log.txt" />              <property name="destination" value="/var/opt/log.txt" />          </properties>          <batchlet ref="sampleBatchlet" />      </step>  </job> |

In this simple JSL file we are executing a Batchlet named "sampleBatchlet" as part of "step1" which takes also two properties. We will use these properties to copy a file from a source to a destination.

# Defining the Batchlet

Here is the Batchlet which is a CDI @Named Bean that collects the properties from the StepContext and uses the Java 7 Files API to copy files from the source to the destination:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.mastertheboss.jberet;    import javax.batch.api.AbstractBatchlet;  import javax.inject.Inject;  import javax.batch.runtime.context.\*;  import javax.inject.Named;  import java.io.\*;  import java.nio.file.Files;    @Named  public class SampleBatchlet extends AbstractBatchlet {      @Inject StepContext stepContext;        @Override      public String process() {           String source = stepContext.getProperties().getProperty("source");           String destination = stepContext.getProperties().getProperty("destination");        try {           Files.copy(new File(source).toPath(), new File(destination).toPath());           System.out.println("File copied!");           return "COMPLETED";       } catch (IOException e) {         e.printStackTrace();       }        return "FAILED";      }    } |

You can trigger the execution of your job with simple main Java class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | package com.mastertheboss.jberet;    import javax.batch.operations.JobOperator;  import javax.batch.operations.JobSecurityException;  import javax.batch.operations.JobStartException;  import javax.batch.runtime.BatchRuntime;    public class Main {        public static void main(String[] args) {          try {              JobOperator jo = BatchRuntime.getJobOperator();                long id = jo.start("simplebatchlet", null);                System.out.println("Batchlet submitted: " + id);              Thread.sleep(5000);            } catch (Exception ex) {              System.out.println("Error submitting Job! " + ex.getMessage());              ex.printStackTrace();          }        }    } |

# Compiling the project

In order to run our project, we need to include in our pom.xml a set of dependencies which include javax.batch Batch API, JBeret core and its related dependencies, Weld container API and its related dependencies:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 | <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <a href="http://maven.apache.org/maven-v4\_0\_0.xsd">http://maven.apache.org/maven-v4\_0\_0.xsd</a>">     <modelVersion>4.0.0</modelVersion>     <groupId>com.mastertheboss.jberet</groupId>     <artifactId>batch-job</artifactId>     <packaging>jar</packaging>     <version>1.0-SNAPSHOT</version>     <name>batch-job</name>     <url><a href="http://maven.apache.org</url>">http://maven.apache.org</url></a>     <repositories>        <repository>           <id>jboss-public-repository-group</id>           <name>JBoss Public Repository Group</name>           <url><a href="http://repository.jboss.org/nexus/content/groups/public/</url>">http://repository.jboss.org/nexus/content/groups/public/</url></a>        </repository>     </repositories>     <dependencies>        <dependency>           <groupId>org.jboss.spec.javax.batch</groupId>           <artifactId>jboss-batch-api\_1.0\_spec</artifactId>           <version>1.0.0.Final</version>        </dependency>        <dependency>           <groupId>org.jberet</groupId>           <artifactId>jberet-core</artifactId>           <version>1.0.2.Final</version>          </dependency>        <dependency>           <groupId>org.jberet</groupId>           <artifactId>jberet-support</artifactId>           <version>1.0.2.Final</version>          </dependency>          <dependency>           <groupId>org.jboss.spec.javax.transaction</groupId>           <artifactId>jboss-transaction-api\_1.2\_spec</artifactId>           <version>1.0.0.Final</version>        </dependency>        <dependency>           <groupId>org.jboss.marshalling</groupId>           <artifactId>jboss-marshalling</artifactId>           <version>1.4.2.Final</version>        </dependency>        <dependency>           <groupId>org.jboss.weld</groupId>           <artifactId>weld-core</artifactId>           <version>2.1.1.Final</version>        </dependency>        <dependency>           <groupId>org.jboss.weld.se</groupId>           <artifactId>weld-se</artifactId>           <version>2.1.1.Final</version>        </dependency>        <dependency>           <groupId>org.jberet</groupId>           <artifactId>jberet-se</artifactId>           <version>1.0.2.Final</version>        </dependency>       </dependencies>     <build>        <plugins>           <plugin>              <groupId>org.codehaus.mojo</groupId>              <artifactId>exec-maven-plugin</artifactId>              <version>1.2.1</version>              <executions>                 <execution>                    <goals>                       <goal>java</goal>                    </goals>                 </execution>              </executions>              <configuration>                 <mainClass>com.mastertheboss.jberet.Main</mainClass>              </configuration>           </plugin>        </plugins>     </build>  </project> |

Optionally you can include also some other dependencies in case you need an XML processor, or the streaming JSON processor:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <dependency>      <groupId>com.fasterxml</groupId>      <artifactId>aalto-xml</artifactId>       <version>0.9.9</version>  </dependency>    <dependency>      <groupId>org.codehaus.woodstox</groupId>      <artifactId>stax2-api</artifactId>      <version>3.1.4</version>  </dependency>    <dependency>      <groupId>com.fasterxml.jackson.core</groupId>      <artifactId>jackson-core</artifactId>      <version>2.4.1</version>  </dependency>  <dependency>      <groupId>com.fasterxml.jackson.dataformat</groupId>      <artifactId>jackson-dataformat-xml</artifactId>      <version>2.4.1</version>  </dependency> |

You can execute your application with:

|  |  |
| --- | --- |
| 1 | mvn clean install exec:java |

After some INFO messages you should check on the Console that:  
Batchlet submitted: 1  
File Copied!

# Configuring JBeret engine

When using Batch Jobs within WildFly container you can configure Jobs persistence and thread pools via the batch subsystem. When running as standalone application you can do it via a file named **jberet.properties** which has to be placed in **src\main\resources** of your Maven project.  
Here follows a sample jberet.properties file:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | # Optional, valid values are jdbc (default), mongodb and in-memory  job-repository-type = jdbc    # Optional, default is jdbc:h2:~/jberet-repo for h2 database as the default job repository DBMS.  # For h2 in-memory database, db-url = jdbc:h2:mem:test;DB\_CLOSE\_DELAY=-1  # For mongodb, db-url includes all the parameters for MongoClientURI, including hosts, ports, username, password,        # Use the target directory to store the DB  db-url = jdbc:h2:./target/jberet-repo  db-user =sa  db-password =sa  db-properties =    # Configured: java.util.concurrent.ThreadPoolExecutor is created with thread-related properties as parameters.  thread-pool-type =    # New tasks are serviced first by creating core threads.  # Required for Configured type.  thread-pool-core-size =    # If all core threads are busy, new tasks are queued.  # int number indicating the size of the work queue. If 0 or negative, a java.util.concurrent.SynchronousQueue is used.  # Required for Configured type.  thread-pool-queue-capacity =    # If queue is full, additional non-core threads are created to service new tasks.  # int indicating the maximum size of the thread pool.  # Required for Configured type.  thread-pool-max-size =    # long number indicating the number of seconds a thread can stay idle.  # Required for Configured type.  thread-pool-keep-alive-time =    # Optional, valid values are true and false, defaults to false.  thread-pool-allow-core-thread-timeout =    # Optional, valid values are true and false, defaults to false.  thread-pool-prestart-all-core-threads =    # Optional, fully-qualified name of a class that implements java.util.concurrent.ThreadFactory.  # This property should not be needed in most cases.  thread-factory =    # Optional, fully-qualified name of a class that implements java.util.concurrent.RejectedExecutionHandler.  # This property should not be needed in most cases.  thread-pool-rejection-policy = |

As you can see, this file largely relies on defaults for many variables like the thread pool. We have anyway applied a change in the **job-repository-type** to persist jobs on a DB (H2 DB). In this case we will need adding the JDBC Driver API to our Maven project as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>           <groupId>com.h2database</groupId>           <artifactId>h2</artifactId>           <version>1.4.178</version>        </dependency> |

That's all! enjoy Batch API using jBeret!

**Acknowledgments**: I'd like to express my gratitude to **Cheng Fang** (JBeret project Lead) for providing useful insights for writing this article