Seam JCR

by George Gastaldi and John Ament

Seam JCR - Introduction

1.1. Introduction

The Seam JCR Module aims to simplify the integration points between JCR implementations and CDI applications.

The Seam JCR module is compatible with JCR 2.0 implementations. It strictly compiles against JCR 2.0. However, test cases are executed against both ModeShape and JackRabbit to ensure compatibility.

1.2. Maven dependency configuration

If you are using *Maven* [http://maven.apache.org/] as your build tool, you can add the following single dependency to your pom.xml file to include Seam JCR:

```
<dependency>
  <groupId>org.jboss.seam.jcr</groupId>
  <artifactId>seam-jcr</artifactId>
  <version>${seam.jcr.version}</version>
</dependency>
```

Tip

Substitute the expression ${seam.jcr.version} with the most recent or appropriate version of Seam JCR. Alternatively, you can create a *Maven user-defined property* to satisfy this substitution so you can centrally manage the version.

Alternatively, you can use the API at compile time and only include the implementation at runtime. This protects you from inadvertantly depending on an implementation class.

```
<dependency>
  <groupId>org.jboss.seam.jcr</groupId>
  <artifactId>seam-jcr-api</artifactId>
  <version>${seam.jcr.version}</version>
  <scope>compile</scope>
</dependency>

<dependency>
  <groupId>org.jboss.seam.jcr</groupId>
```

```
  <artifactId>seam-jcr</artifactId>
  <version>${seam.jcr.version}</version>
  <scope>runtime</scope>
</dependency>
```

In addition to your Seam JCR dependencies, you must also declare a dependency on a JCR Implementation.

Seam JCR - JBoss ModeShape Integration

2.1. ModeShape Integration Installation

In order to activate ModeShape support within your application, you need to include ModeShape on your classpath. At a minimum, the following maven dependencies must be satisfied.

```
<dependency>
   <groupId>org.modeshape</groupId>
   <artifactId>modeshape-jcr</artifactId>
   <version>${modeshape.version}</version>
</dependency>
<dependency>
   <groupId>org.apache.lucene</groupId>
   <artifactId>lucene-core</artifactId>
   <version>${lucene.version}</version>
</dependency>
```

Tip

Substitute ${modeshape.version} for the ModeShape version you are running against. Currently, Seam JCR tests against 2.5.0.Final. In addition, Lucene is required to run ModeShape. Please consult the ModeShape getting stated guide for exact versions.

2.2. Usage

In order to use ModeShape's Repository and Session objects in your application, you must define an injection point using the JcrConfiguration annotation based on ModeShape's required configuration parameters. Please review the ModeShape getting started guide for further details.

```
@Inject @JcrConfiguration(name="org.modeshape.jcr.URL",value="file:path/to/modeshape.xml?repositoryName=MyRepo")
   Repository repository;
```

```
@Inject @JcrConfiguration(name="org.modeshape.jcr.URL",value="file:path/to/modeshape.xml?repositoryName=MyRepo")
Session session;
```

Seam JCR - JackRabbit Integration

3.1. JackRabbit Integration Installation

In order to activate JackRabbit support within your application, you need to include JackRabbit on your classpath. At a minimum, the following maven dependency must be satisfied.

```
<dependency>
   <groupId>org.apache.jackrabbit</groupId>
   <artifactId>jackrabbit-core</artifactId>
   <version>${jackrabbit.version}</version>
</dependency>
```

Tip

Substitute ${jackrabbit.version} for the JackRabbit version you are running against. Currently, Seam JCR tests against 2.2.4. Please review JackRabbit documentation to determine any additional dependencies.

3.2. Usage

In order to use JackRabbit's Repository and Session objects in your application, you must define an injection point using the JcrConfiguration annotation based on JackRabbit's required configuration parameters.

```
@Inject @JcrConfiguration(name="org.apache.jackrabbit.repository.home",value="target")
Repository repository;

@Inject @JcrConfiguration(name="org.apache.jackrabbit.repository.home",value="target")
Session session;
```

Seam JCR - Event Mapping

Seam JCR provides functionality to fire CDI Events based on events found in JCR. The rules of how events are fired are based around the underlying implementation.

Tip

You will not have an active JCR Session during the event firing, a new one will need to be created.

Tip

Some JCR implementations, like Modeshape fires events on a separate thread, so probably you will get errors if your observer method is declared on a @RequestScoped object, for example.

To observe an event, use the @Observes and the additional qualifiers on seam-jcr-api module (Check package org.jboss.seam.jcr.annotations.events). If you need to watch any JCR event, then avoid using any Qualifier at all.

```
import javax.jcr.observation.Event;

public void observeAdded(@Observes @NodeAdded Event evt) {
   // Called when a node is added
}
public void observeAll(@Observes javax.jcr.observation.Event evt) {
   // Called when any node event occurs
}
```

Seam JCR - Object Content Mapping

5.1. What is Object Content Mapping?

Object Content Mapping is a design paradigm, in the same light as ORM (Object Relational Mapping) frameworks such as JPA or Hibernate, where statically typed objects are bound to a storage mechanism, in this case a JCR store. Seam JCR OCM is provided as annotations only on top of entities that are discovered during the CDI Phase ProcessAnnotatedType. In addition, Seam JCR's OCM implementation provides ServiceHandlers for working with entities over JCR.

5.2. Mapping and Conversion Capabilities

The mapping API is very simple and designed to be clean. In order to define an entity, you simply need to use the annotation `org.jboss.seam.jcr.annotations.ocm.JcrNode` to define that this is an entity to map. All fields by default will be mapped to their field names. You can override this behavior by using the annotation `org.jboss.seam.jcr.annotations.ocm.JcrProperty` which will map the property to a different property name. The `JcrProperty` annotation can be placed on both field and getter method. You can define a special property `uuid` of type String that will represent the identifier for the node. This is a sample node mapping:

```
    @JcrNode("nt:unstructured")
public class BasicNode implements java.io.Serializable {
   @JcrProperty("myvalue")
   private String value;
   private String uuid;
   private String lordy;
   public String getValue() {
      return value;
   }
   public void setValue(String value) {
      this.value = value;
   }
   public String getUuid() {
      return uuid;
   }
   public void setUuid(String uuid) {
      this.uuid = uuid;
   }
   @JcrProperty("notaproperty")
   public String getLordy() {
      return lordy;
   }
```

```
    public void setLordy(String lordy) {
        this.lordy = lordy;
    }
}
```

The simplest way to convert entities is to use CDI Events. There are two event objects that can be fired to support parsing, `org.jboss.seam.jcr.ocm.ConvertToNode` and `org.jboss.seam.jcr.ocm.ConvertToObject`. By passing in a node and a pre-constructed object you can convert the full node to object or object to node depending on your need. Here is a sample parsing (from our test cases):

```
@Inject Event<ConvertToObject< objectEvent;
@Inject Event<ConvertToNode< nodeEvent;
....

Node root = session.getRootNode();
Node ocmnode1 = root.addNode("ocmnode1","nt:unstructured");
BasicNode bn = new BasicNode();
bn.setValue("Hello, World!");
bn.setLordy("this was saved.");
nodeEvent.fire(new ConvertToNode(bn,ocmnode1));

Node hello2 = root.getNode("ocmnode1");
BasicNode bn2 = new BasicNode();
objectEvent.fire(new ConvertToObject(hello2,bn2));
```

5.3. JCR Data Access Objects

If you have ever worked with entities, the term DAO should be very familiar to you. Seam JCR OCM supports DAOs in a highly automated fashion. Using annotations and interfaces only, you can automate querying, finds and saving entities into their mapped node types. There are four annotations to support DAOs:

1. `org.jboss.seam.jcr.annotations.ocm.JcrDao` Defines this interface as a DAO interface. The ServiceHandler will be used to process these methods. This annotation should be placed at the interface level.

2. `org.jboss.seam.jcr.annotations.ocm.JcrFind` Defines this method as a find method, loading by identifier. The method should take a single String parameter and return a mapped node type.

3. `org.jboss.seam.jcr.annotations.ocm.JcrQuery` Defines this method as returning a `java.util.List` of mapped entities that can be mapped using the query results. Has properties defining the type to return, query to use, and the query language.

4. `org.jboss.seam.jcr.annotations.ocm.JcrParam` JcrParams are placed on the parameter arguments to a method annotated JcrQuery. Each argument should be a Value object and map based on bind parameters in the query.

Here is a sample definition of an interface, describing the objects that can be used:

```
    import static org.jboss.seam.jcr.ConfigParams.MODESHAPE_URL;

import java.util.List;

import org.jboss.seam.jcr.annotations.JcrConfiguration;
import org.jboss.seam.jcr.annotations.ocm.JcrDao;
import org.jboss.seam.jcr.annotations.ocm.JcrFind;
import org.jboss.seam.jcr.annotations.ocm.JcrQuery;
import org.jboss.seam.jcr.annotations.ocm.JcrSave;
import org.jboss.seam.jcr.test.ocm.BasicNode;

@JcrDao(
   @JcrConfiguration(name = MODESHAPE_URL,
      value = "file:target/test-classes/modeshape.xml?repositoryName=CarRepo")
)
public interface BasicNodeDAO {
   @JcrFind
   public BasicNode findBasicNode(String uuid);
                  @JcrQuery(query="select * from [nt:unstructured]",language="JCR-
SQL2",resultClass=BasicNode.class)
   public List<BasicNode> findAllNodes();
   @JcrSave
   public String save(String path, BasicNode basicNode);
}
```

In this case, we are telling the JcrDao BasicNodeDAO to use the JCR Session based on the annotated JcrConfiguration noted. Since BasicNode is mapped to nt:unstructured, we can map any nt:unstructured to it by calling findAllNodes. We can save a basic node to a given path as well as find based on uuid. The best part is that there is no implementation necessary on your side. You can use this interface as is.

```
    @Inject
  BasicNodeDAO basicDAO;

  ....

    BasicNode bn = new BasicNode();
    bn.setValue("this is my node.");
    String uuid = basicDAO.save("/anypathone",bn);
    System.out.println("The UUID is: "+uuid);

    BasicNode bn2 = basicDAO.findBasicNode(uuid);
                System.out.printf("The original node was %s and the new node is
\n",bn.getValue(), bn2.getValue());

    List<BasicNode> nodes = basicDAO.findAllNodes();
    System.out.println(nodes);
```