Session Serialization Code Walkthrough

When you are using Stateful KieSessions that process realtime data over long periods of time it is possible your session could crash or something could go wrong while the rules are firing. Because of this you need to be able to save your KieSessions so you do not lose all the data in it if something goes wrong. In this series of projects we have a simple example of how to do this.

KieSessions are serializable. You can use them in conjunction with org.kie.api.marshalling. To do this you just need some sort of data storage system to marshal the KieSessions to. In our project we simply marshal to an xml file.

The write function in the ReportServiceSerialized.java class creates and output stream to an xml file and write the KieSession to the file. The session configuration needs to be explicitly serialized as it will not be persisted when you serialize the KieSession. You should do this is you have special session configuration such as a pseudo clock. You will also need access to the KieBase the KieSession was created from in order to marshal and unmarshal the KieSession. The load function does the opposite of the write function. When the KieSession is unmarshalled with a KieSessionConfiguration the configuration is automatically loaded to the KieSession.

drools-example-application/src/main/java/com/redhat/usecase/services/ReportServiceSerialized.java

```
private void write(KieSession session) throws IOException {
    FileOutputStream fos = new FileOutputStream("src/main/java/com/redhat/usecase/services/file.xml");
    ObjectOutputStream oos = new ObjectOutputStream(fos);

    KieSessionConfiguration kieSessionConfiguration = session.getSessionConfiguration();
    oos.writeObject(kieSessionConfiguration);

    Marshaller marshaller = createSerializableMarshaller(brmsUtil.getKieBase());
    marshaller.marshall(fos, session);

}
```

```
private KieSession load() throws ClassNotFoundException, IOException {
    FileInputStream fis = new FileInputStream("src/main/java/com/redhat/usecase/services/file.xml");
    ObjectInputStream ois = new ObjectInputStream(fis);

    KieSessionConfiguration kieSessionConfiguration = (KieSessionConfiguration) ois.readObject();

    Marshaller marshaller = createSerializableMarshaller(brmsUtil.getKieBase());
    return marshaller.unmarshall(fis, kieSessionConfiguration, null);

}
```

```
private Marshaller createSerializableMarshaller(KieBase kBase) {
    ObjectMarshallingStrategyAcceptor acceptor = MarshallerFactory.newClassFilterAcceptor(new String[] { "*.*" });
    ObjectMarshallingStrategy strategy = MarshallerFactory.newSerializeMarshallingStrategy(acceptor);
    Marshaller marshaller = MarshallerFactory.newMarshaller(kBase, new ObjectMarshallingStrategy[] { strategy });
    return marshaller;
}
```

How you marshal and unmarshal your session is up to you. In the class we marshal the session once it is created then we marshal it again right before we fire the rules. If something did go wrong when the rules are firing then you could unmarshal the session and fire all rules again. There are many ways to build upon this to give your application fault tolerance.

drools-example-application/src/main/java/com/redhat/usecase/services/ReportServiceSerialized.java

```
createSession();
KieSession session = load();

for (TradeEvent trade : inputTrades) {
    List<Command> list = new ArrayList<Command>();
    System.out.println("TRADE: " + trade);
    // session.addEventListener(listener);

    list.add(CommandFactory.newInsert(trade));
    session.insert(trade);

}
write(session);

session.fireAllRules();
```

```
 private void createSession() throws IOException {
     brmsUtil = new BRMSUtil("serialize-session-example", "stream");
     KieSession session = brmsUtil.getStatefulSession("realtime");
     write(session);
 }
```