Intro

I'm writting this tutorial as a first attempt to share my JBPM knowledge and progress with other people.

What I'm trying to achieve is:
- JBPM deployed as a service archive on JBoss
- JBPM configured to run in a CMT JTA environment with separate databases for the JBPM engine and the custom JbpmFirst application
- EJB3 layer for developing the business logic in the JbpmFirst application
- a JBPM abstraction layer between the EJB3 layer and the JBPM engine
- JSF for the web layer

I will skip the default steps of downloading and installing the tools we use: JDK 6, Ant 1.7, JBoss AS 5.1.0.GA, Eclipse Galileo for EE developers, JBoss Tools, JBPM 4.3, Oracle 10g. You can find all these and related tutorials on setting them up on the first search, so I will focus on our subject.

This tutorial was performed on a Windows 7 machine, but feel free to try it on any operating system.

Step 1: Database setup

Create jbpm/jbpm and sedd/sedd users in Oracle. Give them dba permissions, since they will be used by Hibernate to create/change/delete tables in your databases.

Connect to Oracle using the jbpm user then create and update the database using the scripts found in D:\Viorel\jbpm-4.3\install\src\db. You can also do this work by opening a command prompt and going to the D:\Viorel\jbpm-4.3\install folder. In here type 'ant create.jbpm.schema' and then 'ant upgrade.jbpm.schema'.

If you want to use the jbpm-console, I suggest loading the example identities also. Open a database connection with the jbpm user and run the script D:\Viorel\jbpm-4.3\install\src\demo\example.identities.sql or just type 'ant load.example.identities' in a command propt under the D:\Viorel\jbpm-4.3\install folder.

Step 2: Install JBPM

Extract JBPM and edit D:\Viorel\jbpm-4.3\install\jdbc\oracle.properties to reflect your database setup

```
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@localhost:1521:xe
jdbc.username=jbpm
```

jdbc.password=jbpm

Then create .jbpm4\build.properties file in your home folder(C:\Users\Viorel). Inside, setup your JBPM configuration to reflect your local settings

database=oracle
tx=standalone
jboss.version=5.1.0.GA
jboss.parent.dir=D:/Viorel/jbpmtests

NOTES:
1. Make sure you use slash / instead of backslash \ even on Windows machines.
2. JBPM is not allowed to redistribute the Oracle JDBC driver so you'll have to put it by hand in the D:\Viorel\jbpm-4.3\lib folder. You can find the ojdbc14.jar in your local installation of Oracle D:\Viorel\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib

Now open a command prompt and go to the D:\Viorel\jbpm-4.3\install folder. In here type 'ant -p' to see all the available targets and their description. Just run the following targets:

ant clean.cfg.dir
ant create.cfg
ant install.jbpm.into.jboss

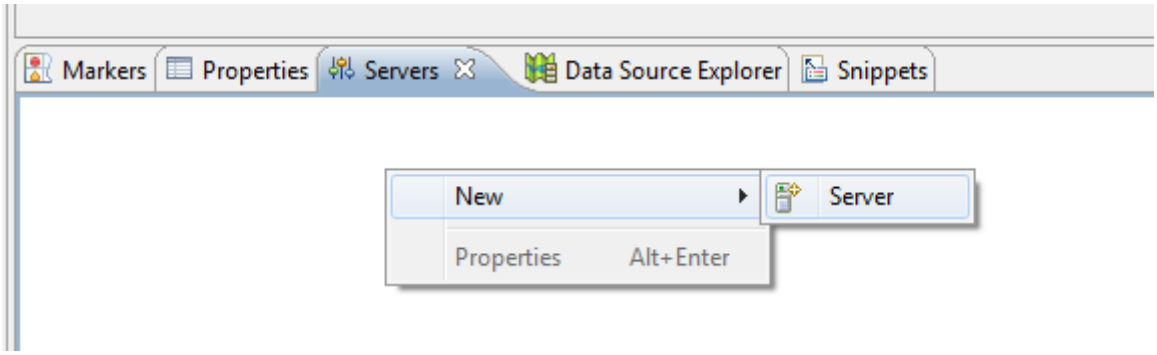
If you see success, you are set. So far, so great and many thanks to the JBPM team.

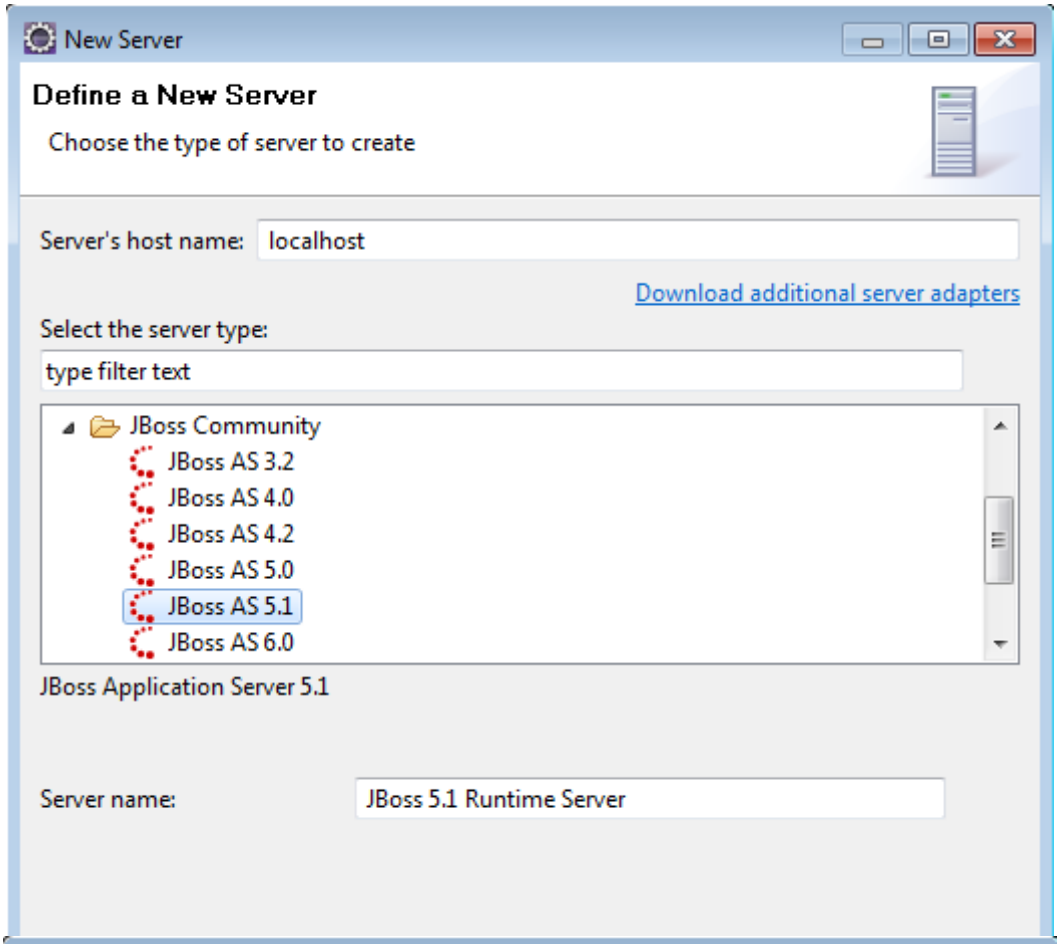
NOTE: You may have to manually change some content in the data source file deployed on JBoss D:\Viorel\jbpmtests\jboss-5.1.0.GA\server\default\deploy\jbpm\jbpm-oracle-ds.xml. This is how it looks on my machine

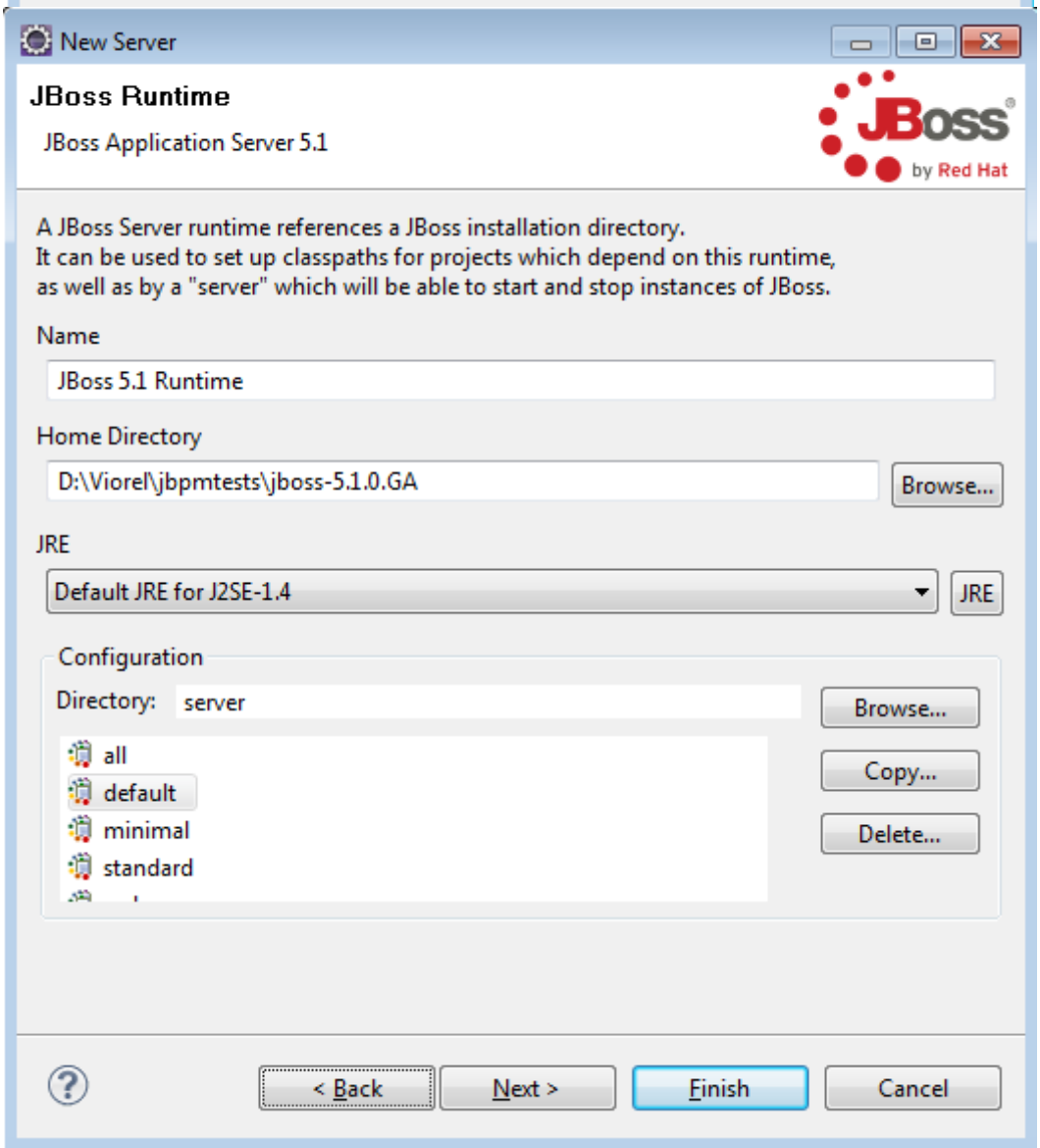
```
<datasources>
  <xa-datasource>
    <jndi-name>JbpmDS</jndi-name>
    <track-connection-by-tx/>
    <!-- uncomment to enable interleaving <interleaving/> -->
    <isSameRM-override-value>false</isSameRM-override-value>
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-dataso
    <xa-datasource-property name="URL">jdbc:oracle:thin:@localhost:1521:xe</
    <xa-datasource-property name="User">jbpm</xa-datasource-property>
    <xa-datasource-property name="Password">jbpm</xa-datasource-property>
    <!-- Uses the pingDatabase method to check a connection is still valid b
    <!--valid-connection-checker-class-name>org.jboss.resource.adapter.jdbc.
    <!-- Checks the Oracle error codes and messages for fatal errors -->
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.Orac
    <!-- Oracles XA datasource cannot reuse a connection outside a transacti
    <no-tx-separate-pools/>

      <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (opti
      <metadata>
        <type-mapping>Oracle9i</type-mapping>
      </metadata>
  </xa-datasource>
```

Step 3. Configure Eclipse

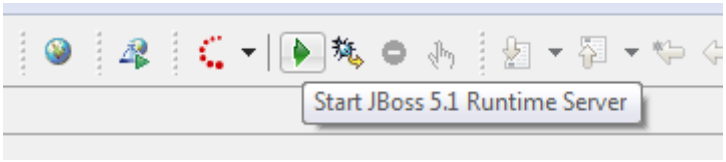
If you have successfully installed JBoss Tools, you can now create a JBoss 5.1 server runtime pointing to the location where you've installed JBPM: D:\Viorel\jbpmtests\jboss-5.1.0.GA




New Server
Define a New Server
Choose the type of server to create
Server's host name:
localhost
Download additional server adapters
Select the server type:
type filter text
JBoss Community
JBoss AS 3.2
JBoss AS 4.0
JBoss AS 4.2
JBoss AS 5.0
JBoss AS 5.1
JBoss AS 6.0
JBoss Application Server 5.1
Server name:
JBoss 5.1 Runtime Server


New Server
JBoss Runtime
JBoss Application Server 5.1
JBoss by Red Hat
A JBoss Server runtime references a JBoss installation directory.
It can be used to set up classpaths for projects which depend on this runtime,
as well as by a "server" which will be able to start and stop instances of JBoss.
Name
JBoss 5.1 Runtime
Home Directory
D:\Viorel\jbpmtests\jboss-5.1.0.GA
Browse...
JRE
Default JRE for J2SE-1.4
JRE
Configuration
Directory:
server
Browse...
all
default
minimal
standard
Copy...
Delete...
< Back
Next >
Finish
Cancel

Check to see if JBPM was deployed successfully.



You should see Hibernate mapping messages

```
[Environment] Hibernate 3.3.1.GA
[Environment] hibernate.properties not found
[Environment] Bytecode provider name : javassist
[Environment] using JDK 1.4 java.sql.Timestamp handling
[Configuration] configuring from resource: jbpm.hibernate.cfg.xml
[Configuration] Configuration resource: jbpm.hibernate.cfg.xml
[Configuration] Reading mappings from resource : jbpm.repository.hbm.
[HbmBinder] Mapping class: org.jbpm.pvm.internal.repository.Deploymen
[HbmBinder] Mapping class: org.jbpm.pvm.internal.repository.Deploymen
[HbmBinder] Mapping class: org.jbpm.pvm.internal.id.PropertyImpl -> J
[Configuration] Reading mappings from resource : jbpm.execution.hbm.x
[HbmBinder] Mapping class: org.jbpm.pvm.internal.model.ExecutionImpl
[HbmBinder] Mapping class: org.jbpm.pvm.internal.type.Variable -> JBP
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Blo
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Dat
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Dou
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Hib
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Hib
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Lon
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Nul
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Str
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.type.variable.Tex
[HbmBinder] Mapping class: org.jbpm.pvm.internal.lob.Lob -> JBPM4_LOB
[HbmBinder] Mapping class: org.jbpm.pvm.internal.job.JobImpl -> JBPM4
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.job.MessageImpl -
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.model.op.ExecuteA
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.model.op.ExecuteE
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.job.CommandMessag
[HbmBinder] Mapping subclass: org.jbpm.pvm.internal.job.TimerImpl ->
[Configuration] Reading mappings from resource : jbpm.history.hbm.xml
```

JNDI bindings

```
[NamingHelper] JNDI InitialContext properties:{}
[CheckDbCmd] jBPM version info: library[4.3], schema[null]
[JbpmService] JbpmService started
[ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=C
[ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=D
[EjbDeployer] installing bean: ejb/#CommandExecutor,uid7774800
[EjbDeployer]   with dependencies:
[EjbDeployer]   and supplies:
[EjbDeployer]        jndi:java:jbpm/CommandExecutor
[EjbDeployer]        jndi:jbpm/CommandExecutor
[EjbDeployer] installing bean: ejb/#CommandReceiver,uid12820137
[EjbDeployer]   with dependencies:
[EjbDeployer]   and supplies:
[EjbDeployer]        jndi:null
[EjbDeployer] installing bean: ejb/#Timer,uid15046604
[EjbDeployer]   with dependencies:
[EjbDeployer]   and supplies:
[EjbDeployer]        jndi:java:jbpm/Timer
[EjbDeployer]        jndi:Timer
[EjbModule] Deploying CommandExecutor
[EjbModule] EJB configured to bypass security. Please verify if this is intend
[EjbModule] Deploying CommandReceiver
[EjbModule] EJB configured to bypass security. Please verify if this is intend
[EjbModule] Deploying Timer
[EjbModule] EJB configured to bypass security. Please verify if this is intend
[BaseLocalProxyFactory] Bound EJB LocalHome 'CommandExecutor' to jndi 'java:jb
[ProxyFactory] Bound EJB Home 'CommandExecutor' to jndi 'jbpm/CommandExecutor'
[BaseLocalProxyFactory] Bound EJB LocalHome 'Timer' to jndi 'java:jbpm/Timer'
[JBossASKernel] Created KernelDeployment for: profileservice-secured.jar
```

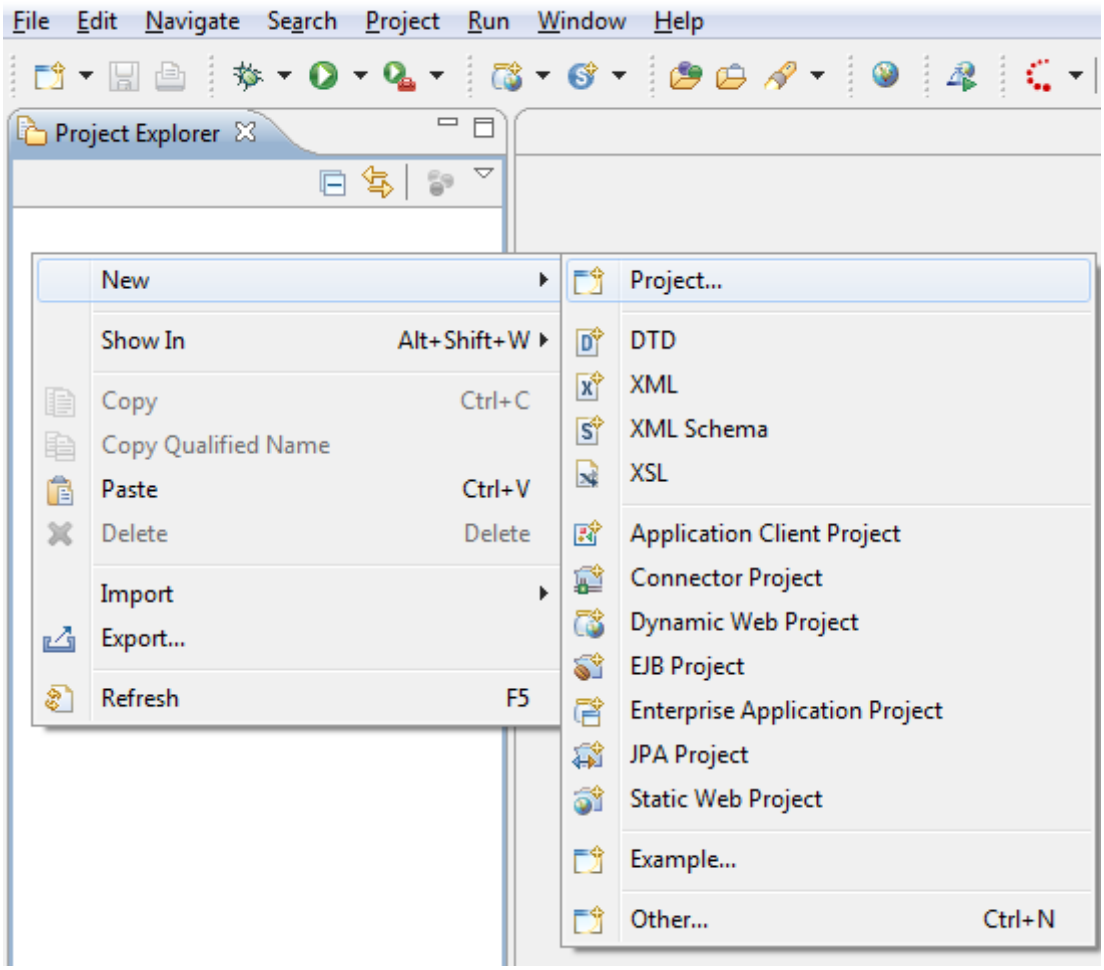
and something related to process engine

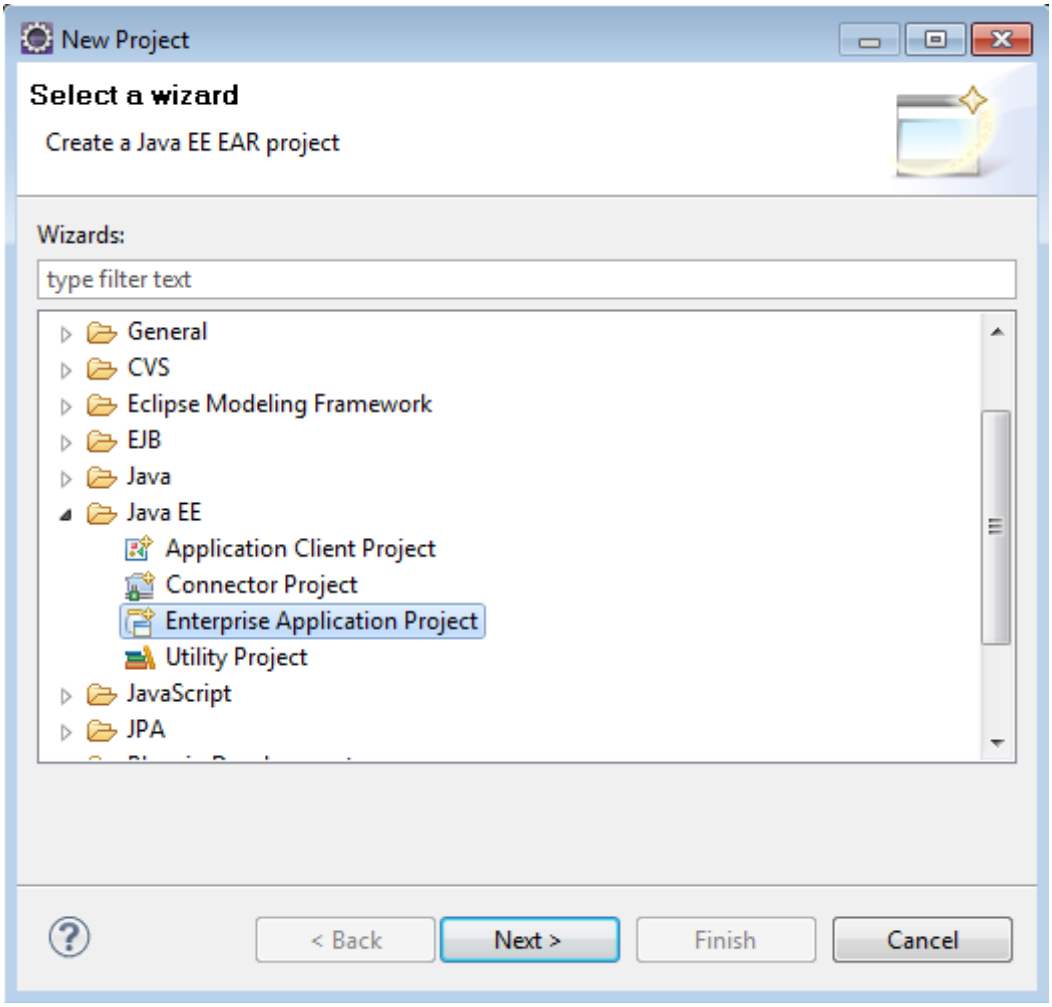
```
 Starting Coyote AJP/1.3 on ajp-localhost%2F127.0.0.1-8009
JBoss (Microcontainer) [5.1.0.GA (build: SVNTag=JBoss_5_1_0_GA date=2009052210
ertyType] The value of the structure property is a string
ertyType] The value of the structure property is a string
ertyType] The value of the structure property is a string
ertyType] The value of the structure property is a string
 Service created: org.eclipse.birt.report.engine.api.impl.ReportEngine@10ab825
```

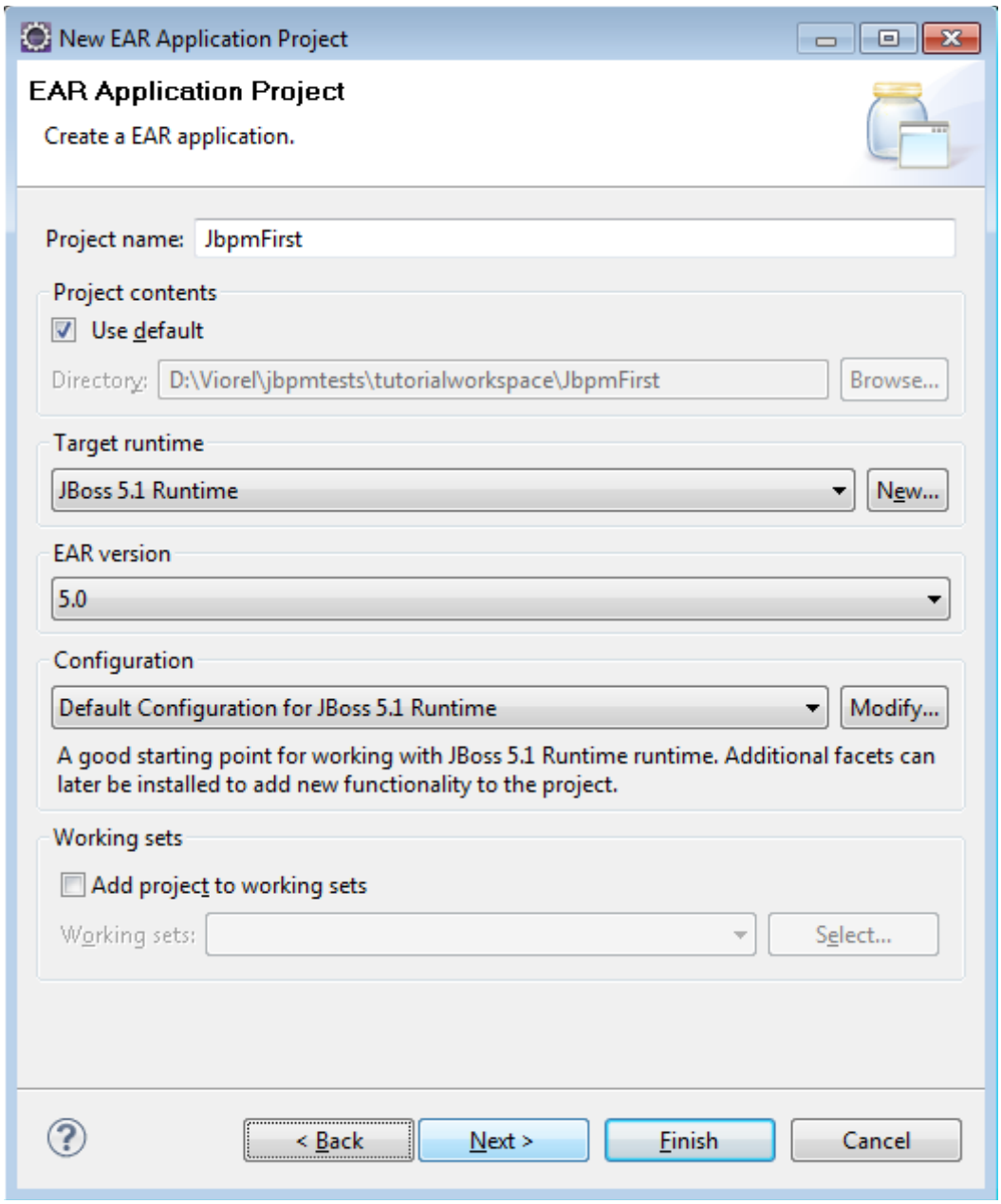
and, of course no exception and no error message; warnings are OK for the purpose of this tutorial.

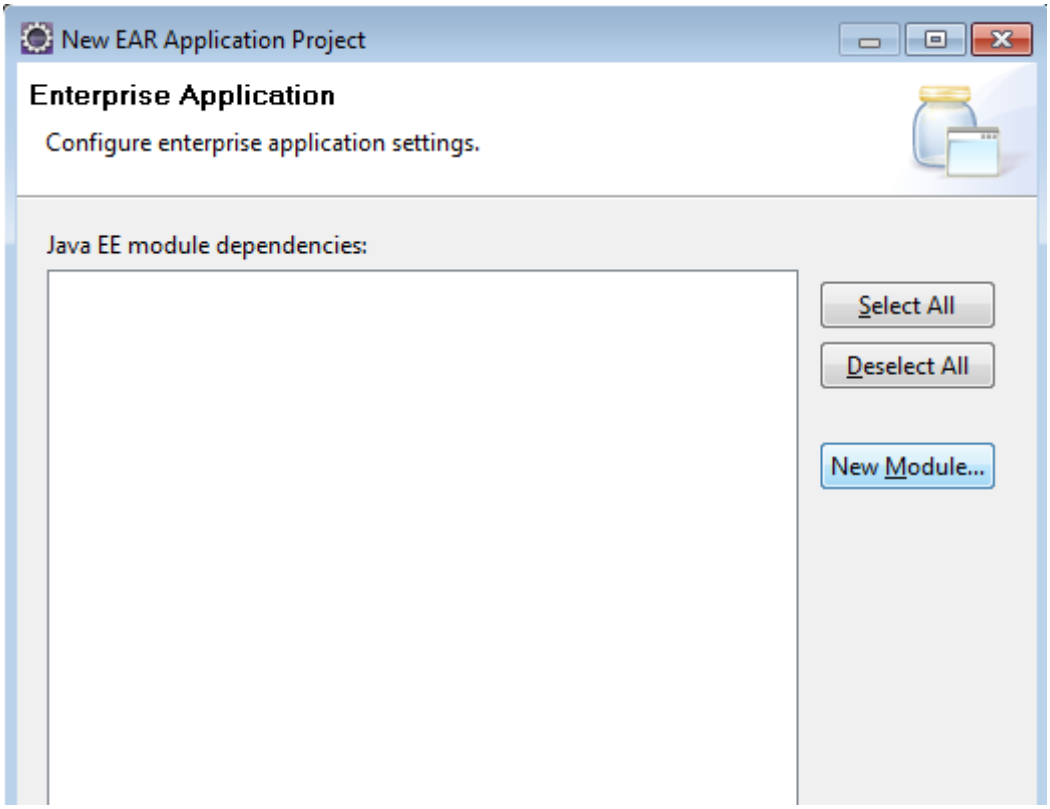
NOTE: Normally there should be a JNDI binding message for the ProcessEngine but there isn't one on my machine. http://community.jboss.org/message/531372#531372

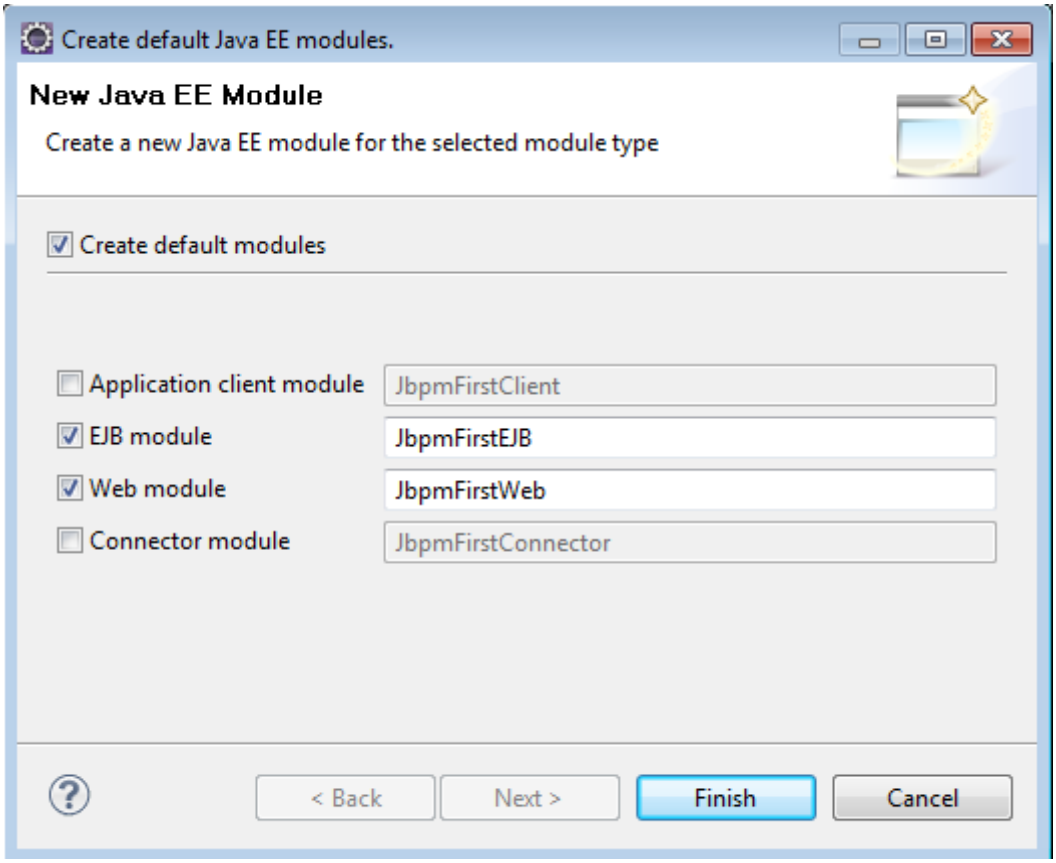
Step 4. Create projects

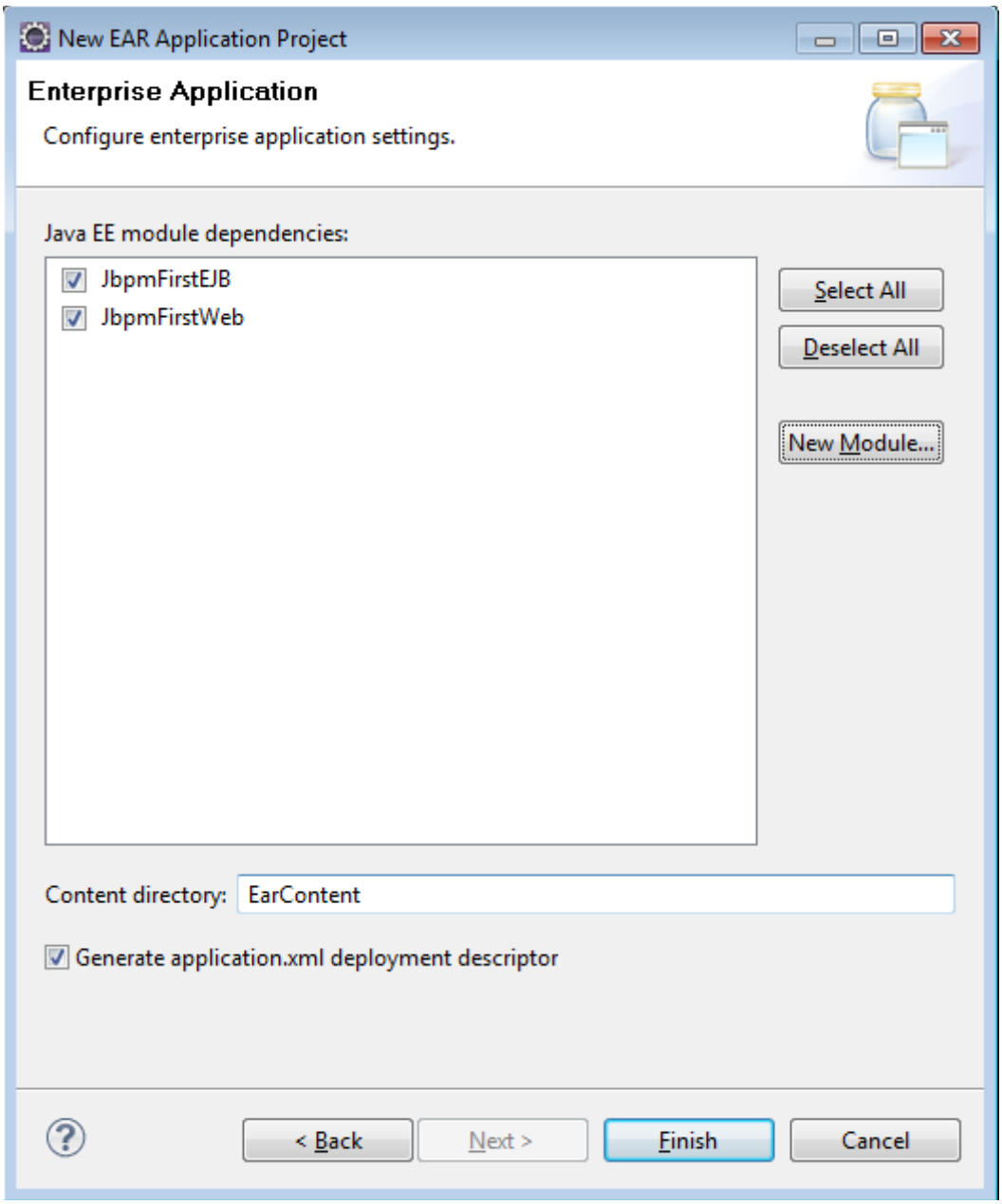

File Edit Navigate Search Project Run Window Help
Project Explorer
New
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Import
Export...
Refresh F5
Project...
DTD
XML
XML Schema
XSL
Application Client Project
Connector Project
Dynamic Web Project
EJB Project
Enterprise Application Project
JPA Project
Static Web Project
Example...
Other... Ctrl+N

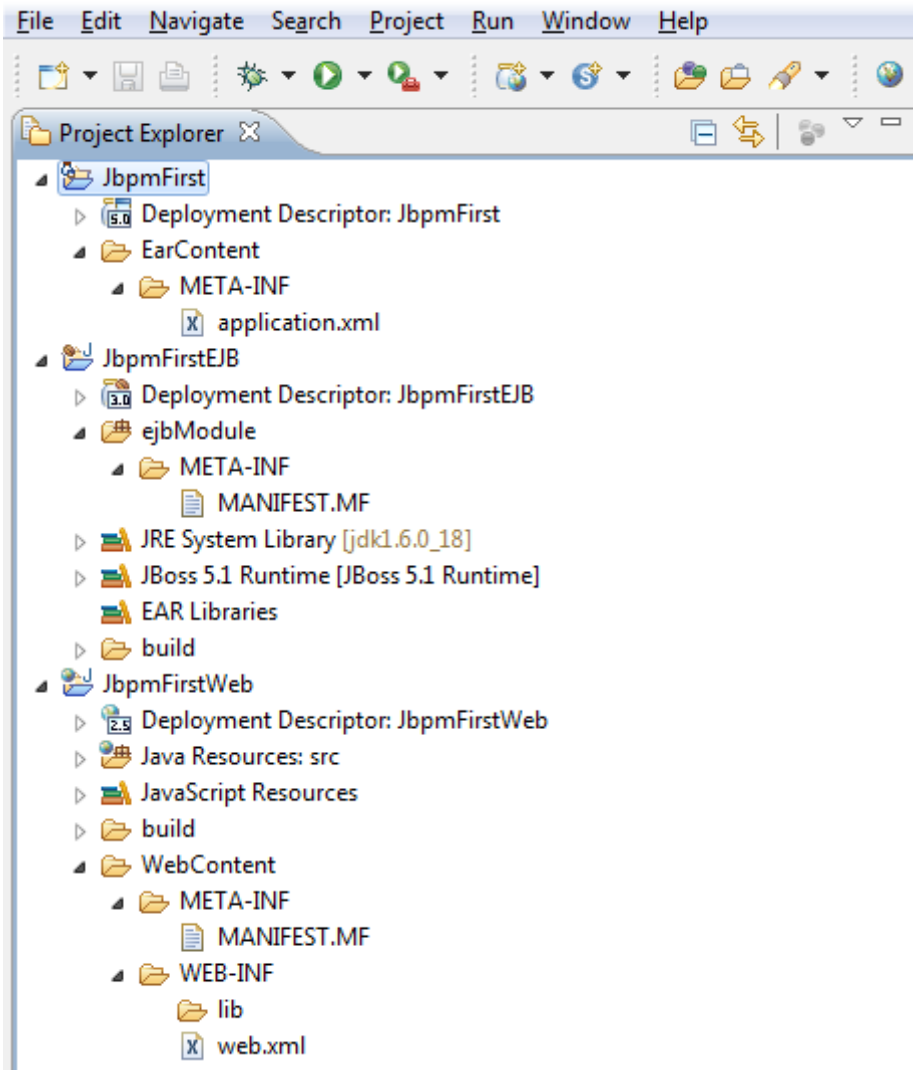

New Project
Select a wizard
Create a Java EE EAR project
Wizards:
type filter text
General
CVS
Eclipse Modeling Framework
EJB
Java
Java EE
Application Client Project
Connector Project
Enterprise Application Project
Utility Project
JavaScript
JPA
< Back
Next >
Finish
Cancel


New EAR Application Project
EAR Application Project
Create a EAR application.
Project name: JbpmFirst
Project contents
Use default
Directory: D:\Viorel\jbpmtests\tutorialworkspace\JbpmFirst
Browse...
Target runtime
JBoss 5.1 Runtime
New...
EAR version
5.0
Configuration
Default Configuration for JBoss 5.1 Runtime
Modify...
A good starting point for working with JBoss 5.1 Runtime runtime. Additional facets can later be installed to add new functionality to the project.
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel

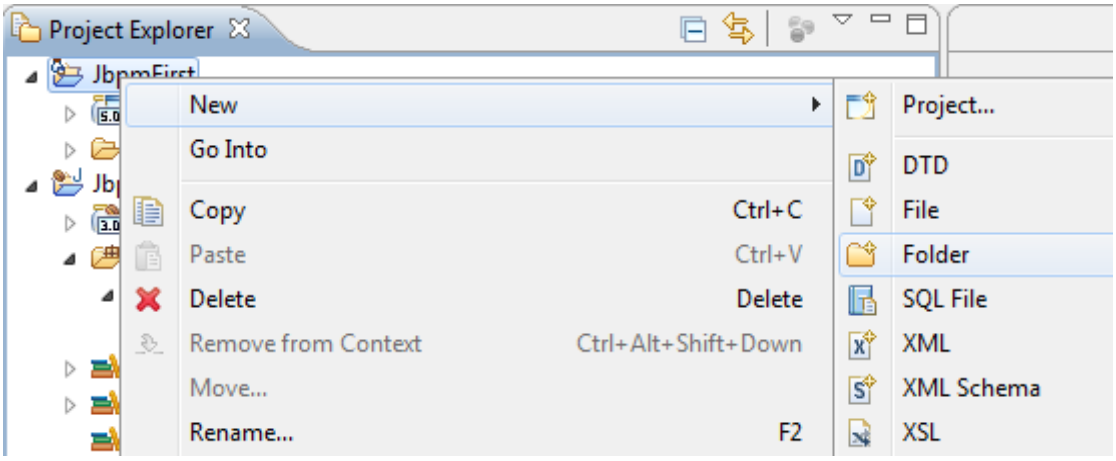

New EAR Application Project
Enterprise Application
Configure enterprise application settings.
Java EE module dependencies:
Select All
Deselect All
New Module...

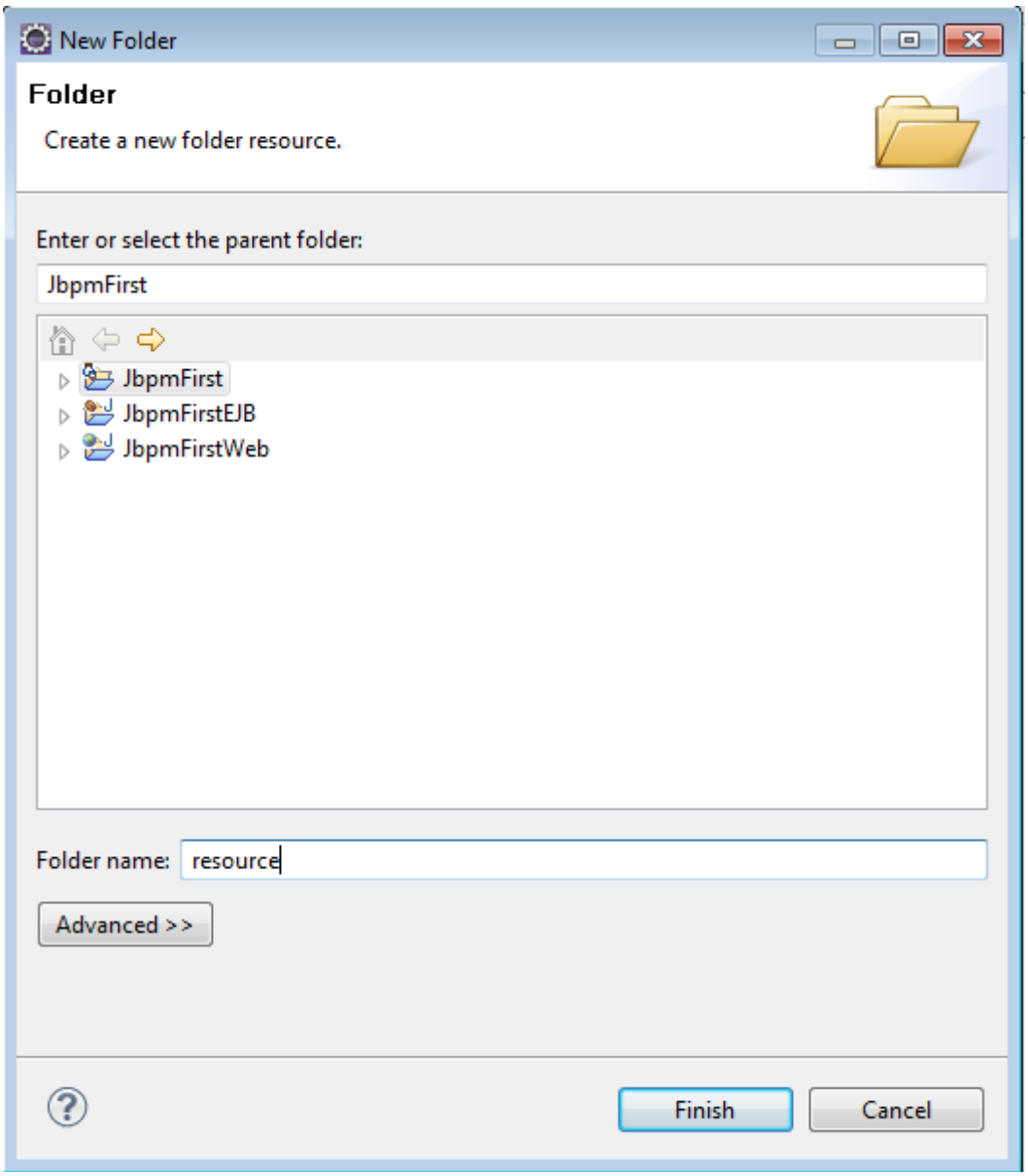

Create default Java EE modules.
New Java EE Module
Create a new Java EE module for the selected module type
Create default modules
Application client module
JbpmFirstClient
EJB module
JbpmFirstEJB
Web module
JbpmFirstWeb
Connector module
JbpmFirstConnector
< Back
Next >
Finish
Cancel

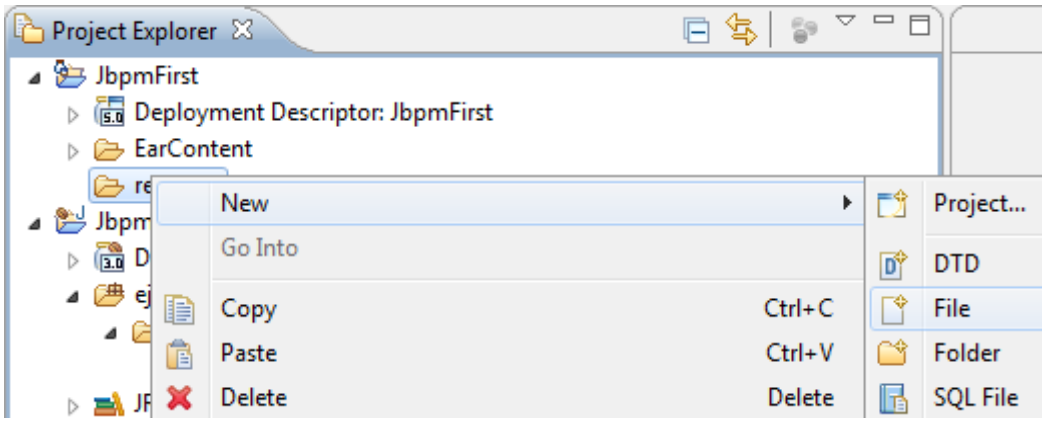

New EAR Application Project
Enterprise Application
Configure enterprise application settings.
Java EE module dependencies:
JbpmFirstEJB
JbpmFirstWeb
Select All
Deselect All
New Module...
Content directory:
EarContent
Generate application.xml deployment descriptor
< Back
Next >
Finish
Cancel

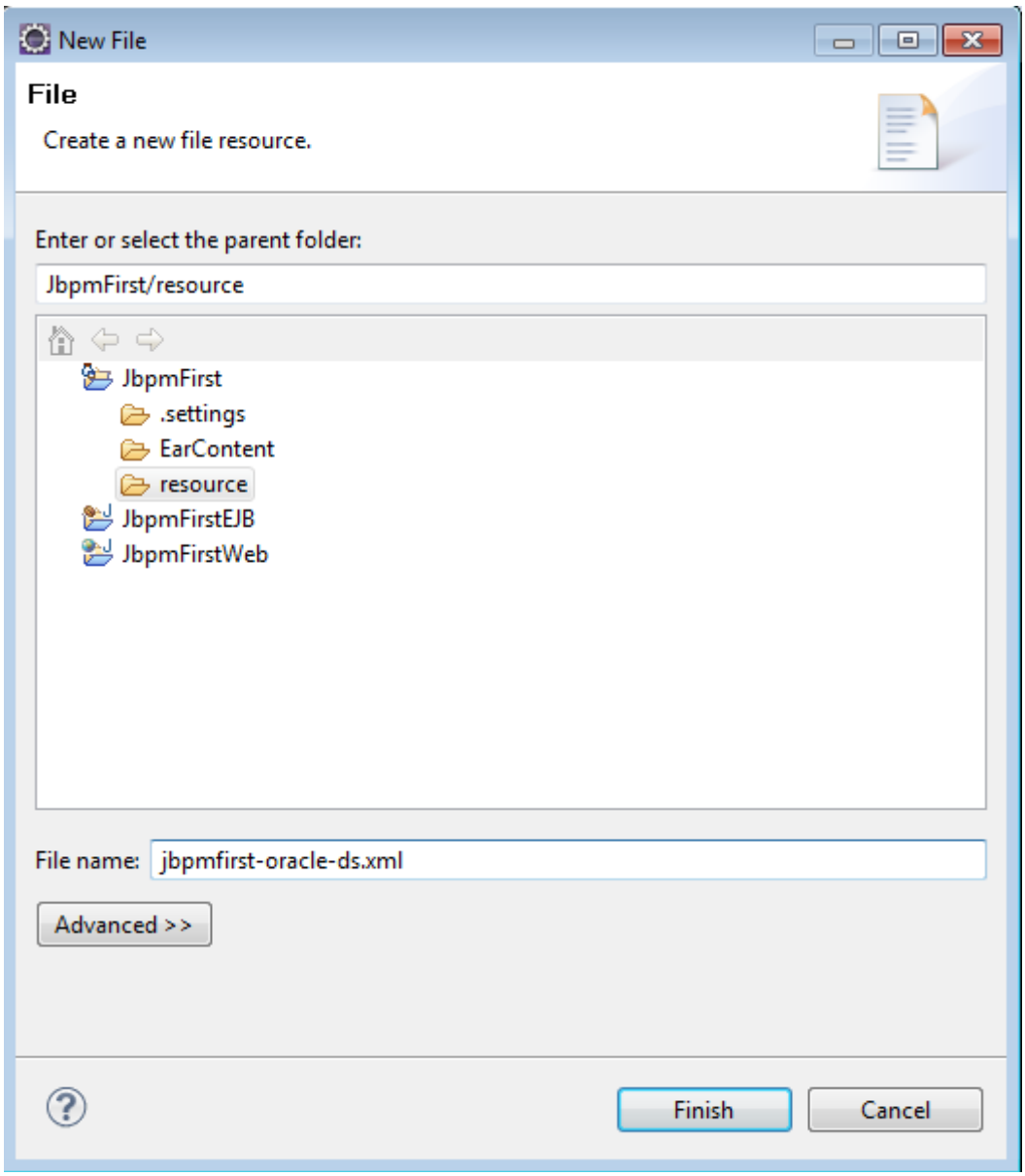


Now let's create a data source file for our application. Currently there is no way to automatically deploy a data source file to JBoss, so we'll have to just create it in our project and deploy it manually.




New Folder
Folder
Create a new folder resource.
Enter or select the parent folder:
JbpmFirst
JbpmFirst
JbpmFirstEJB
JbpmFirstWeb
Folder name: resource
Advanced >>
Finish
Cancel


Project Explorer
JbpmFirst
Deployment Descriptor: JbpmFirst
EarContent
New
Go Into
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Project...
DTD
File
Folder
SQL File



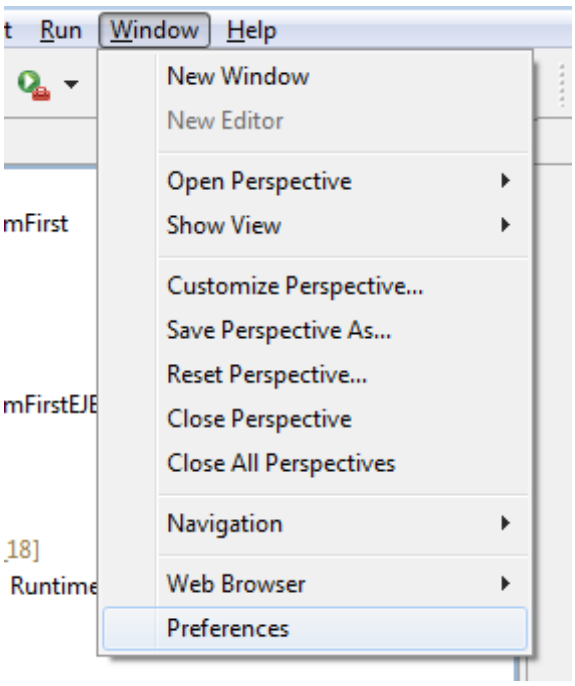
This data source is similar to the one configured for the JBPM engine so it's OK to copy that content, change the name and the user and remove the <mbean> tag.

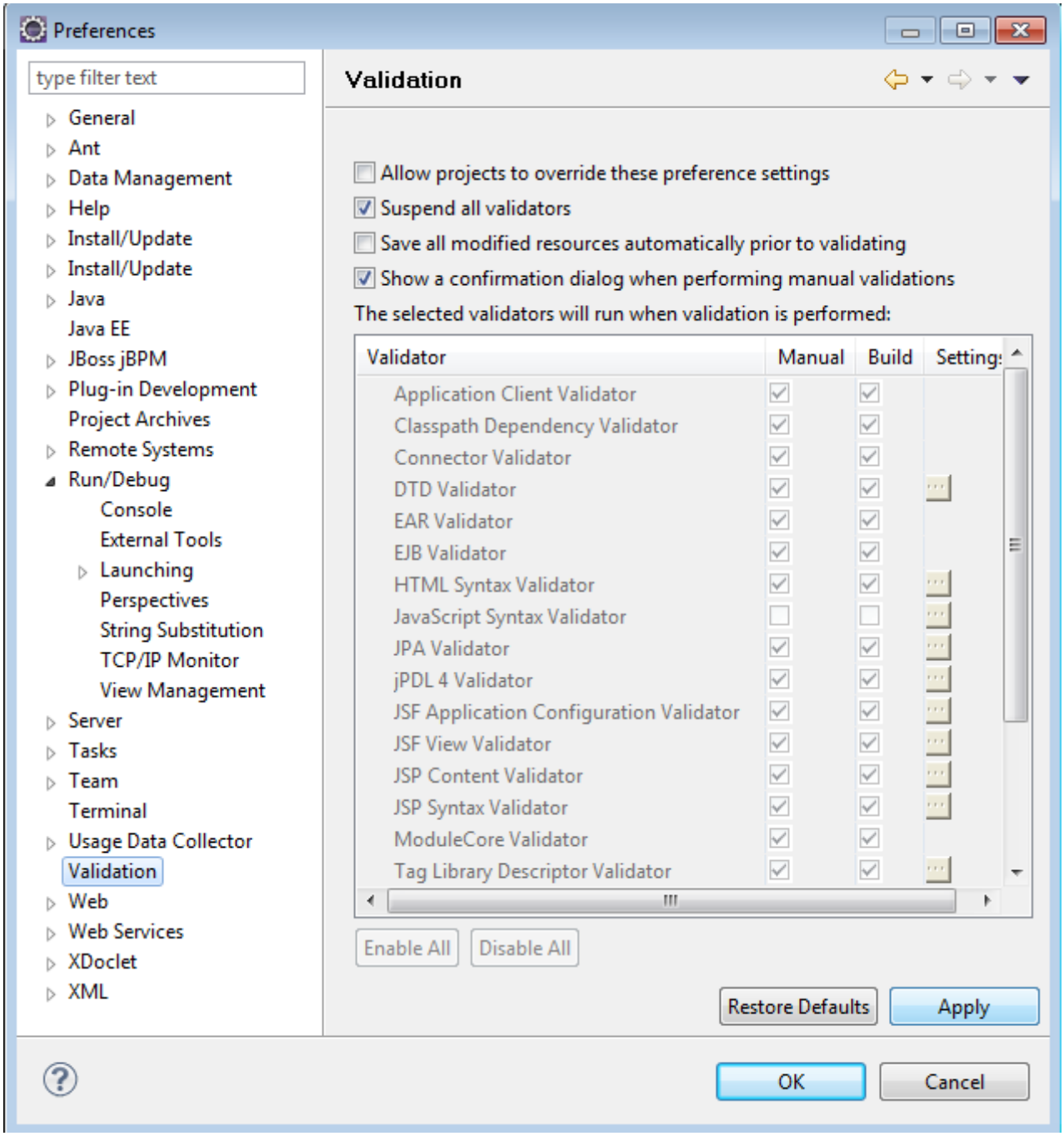
jbpmfirst-oracle-ds.xml

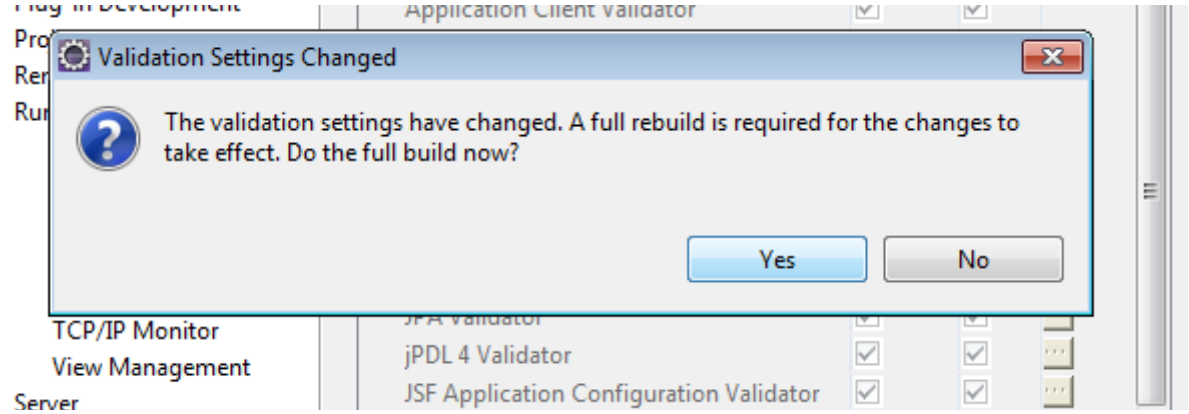
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE datasources
    PUBLIC "-//JBoss//DTD JBOSS JCA Config 5.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-ds_5_0.dtd">

<datasources>
  <xa-datasource>
    <jndi-name>SeddJbpmDS</jndi-name>
    <track-connection-by-tx/>
    <!-- uncomment to enable interleaving <interleaving/> -->
    <isSameRM-override-value>false</isSameRM-override-value>
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasour
    <xa-datasource-property name="URL">jdbc:oracle:thin:@localhost:1521:xe</xa
    <xa-datasource-property name="User">sedd</xa-datasource-property>
    <xa-datasource-property name="Password">sedd</xa-datasource-property>
    <!-- Uses the pingDatabase method to check a connection is still valid bef
    <!--valid-connection-checker-class-name>org.jboss.resource.adapter.jdbc.ve
    <!-- Checks the Oracle error codes and messages for fatal errors -->
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.Oracle
    <!-- Oracles XA datasource cannot reuse a connection outside a transaction
    <no-tx-separate-pools/>

      <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (option
      <metadata>
         <type-mapping>Oracle9i</type-mapping>
      </metadata>
  </xa-datasource>
</datasources>
```

TODO: investigate if this configuration is enough to make the two data sources work together in a two phase commit way on the same transaction managed by the EJB container.

It's time to disable Eclipse validation to avoid the annoying XML errors.





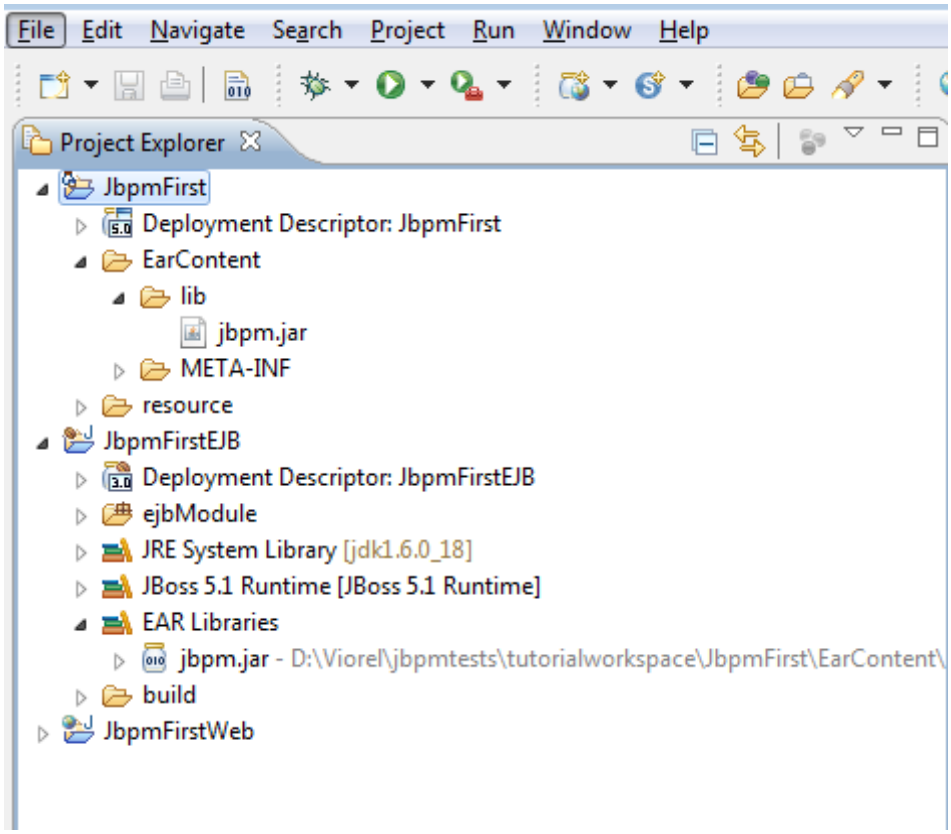


Now if you right-click the JbpmFirst/resources folder and select validate you should see no more errors.

Until there will be a deployment tool available for data source files, you are stuck with deploying it manually to JBoss, so just copy the jbpmfirst-ora-ds.xml file to D:\Viorel\jbpmtests\jboss-5.1.0.GA\server\default\deploy. You should see the following message appended to your console

```
Bound ConnectionManager 'jboss.jca:service=DataSourceBinding,name=SeddJbpmDS'

                        to JNDI name 'java:SeddJbpmDS'
```

The final configuration is adding the jbpm.jar file to our application's classpath. Just create a lib folder inside JbpmFirst/EarContent and copy D:\Viorel\jbpm-4.3\jbpm.jar file to it.



If everything is OK you will see the jbpm.jar library already referenced in the EJB classpath. Now we're all set for development.

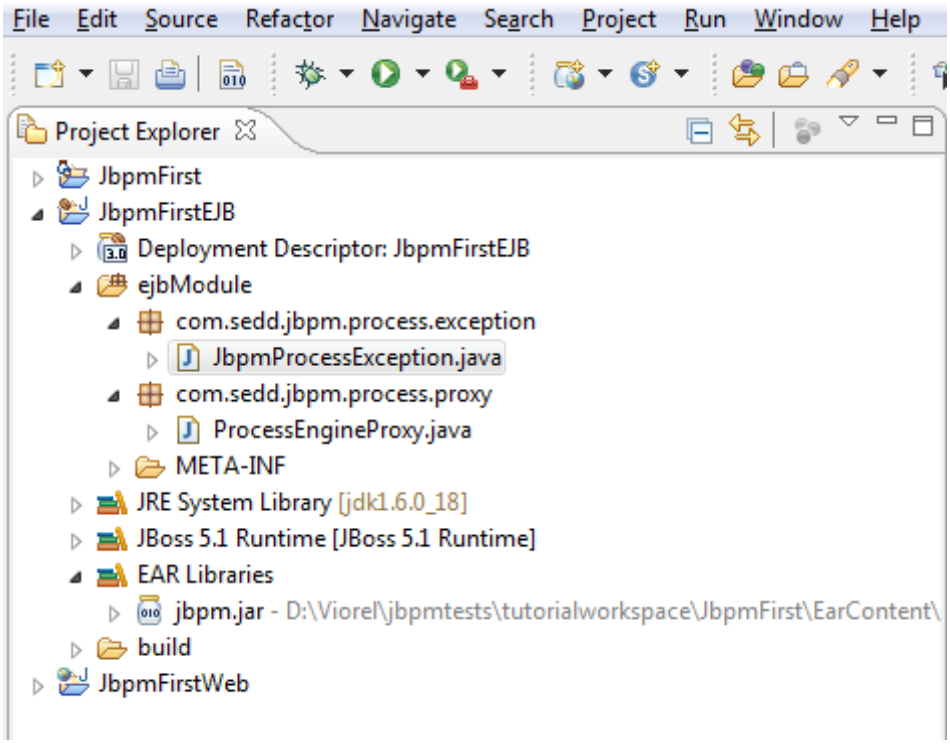
Step 5. Coding

I won't go into much Eclipse details for this part.

The first thing we want to do is create some sort of abstraction layer between our session beans and the JBPM related stuff, so that the application code remains decoupled from the JBPM details. This way we can choose to change JBPM with other BPM engine or upgrade to newer versions of JBPM with minimum amount of changes. The reason for having an abstraction layer between the application code and the JBPM engine is evident if you only look at the amount of changes between JBPM 3.x and JBPM 4.x: it has been completely rewritten and. So, let's see how we can indirect

calls from session beans to the JBPM process engine.

Let's keep our abstraction layer stuff in the com.sedd.jbpm.process java package. First we should create a proxy for the services exposed by JBPM.



package com.sedd.jbpm.process.proxy;

```
import com.sedd.jbpm.process.exception.JbpmProcessException;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.jbpm.api.ExecutionService;
import org.jbpm.api.HistoryService;
import org.jbpm.api.ManagementService;
import org.jbpm.api.ProcessEngine;
import org.jbpm.api.RepositoryService;
import org.jbpm.api.TaskService;

public class ProcessEngineProxy {

	private static final String JNDI_NAME = "java:/ProcessEngine";

	private static ProcessEngine processEngine;

	private ProcessEngineProxy() {
	}

	public static ProcessEngine getProcessEngine() {
		if (ProcessEngineProxy.processEngine == null) {
			try {
				InitialContext ctx = new InitialContext();
				ProcessEngineProxy.processEngine = (ProcessEngine) ctx
						.lookup(JNDI_NAME);
			} catch (NamingException e) {
				throw new JbpmProcessException(
						"Could not find Process Engine.", e);
```

```
                }
            }
            return ProcessEngineProxy.processEngine;
        }

        public static RepositoryService getRepositoryService() {
            return ProcessEngineProxy.getProcessEngine().getRepositoryService();
        }

        public static ExecutionService getExecutionService() {
            return ProcessEngineProxy.getProcessEngine().getExecutionService();
        }

        public static TaskService getTaskService() {
            return ProcessEngineProxy.getProcessEngine().getTaskService();
        }

        public static HistoryService getHistoryService() {
            return ProcessEngineProxy.getProcessEngine().getHistoryService();
        }

        public static ManagementService getManagementService() {
            return ProcessEngineProxy.getProcessEngine().getManagementService();
        }

}
```

The exception class is straight forward.

```
package com.sedd.jbpm.process.exception;

public class JbpmProcessException extends RuntimeException {

        private static final long serialVersionUID = 5450926160398675485L;

        public JbpmProcessException() {
            super();
        }

        public JbpmProcessException(String arg0, Throwable arg1) {
            super(arg0, arg1);
        }

        public JbpmProcessException(String arg0) {
            super(arg0);
        }

        public JbpmProcessException(Throwable arg0) {
            super(arg0);
        }

}
```

For now we can stop with the generic stuff. We will continue with the process specific details and add more generic stuff to the abstraction layer as soon as we encounter repeatable work.

Let's take a business process example and discuss on it. The following process is a modified version of the state choice example that comes with JBPM.

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="DocumentApproval" xmlns="http://jbpm.org/4.3/jpdl">

	<start name="start">
		<transition to="check"/>
	</start>

	<state name="check">
		<transition name="reject" to="rejected"/>
		<transition name="accept" to="submit"/>
	</state>

	<state name="submit">
		<transition to="done"/>
	</state>

	<end name="done">
	</end>

	<end-error name="rejected">
	</end-error>

</process>
```

When this process is started, it executes the start tag which moves the process execution to the wait state called check. When it enters this state, the process persists it's state to the database and waits for an external signal to trigger the next move. The external signal must come with a tag on it to choose from the two transitions. If we signal the process with “reject”, then the process execution moves to the rejected state which is an error state and ends the process. If we signal the process with “accept”, then the process moves to the next wait state called submit. We can signal this state with no tag, and when we do so the process execution moves to the next state called done which is an end state, so the process stops.

First let's see the abstraction code that we use on top of this process and afterward we'll see how we can deploy this process. We should create a specific package for each deployed process to obtain an intuitive business oriented code grouping. Thus create the com.sedd.jbpm.process.documentapproval package and inside create the following class

```
package com.sedd.jbpm.process.documentapproval;

import org.jbpm.api.Execution;
import org.jbpm.api.ProcessInstance;

import com.sedd.jbpm.process.exception.JbpmProcessException;
import com.sedd.jbpm.process.proxy.ProcessEngineProxy;

public class DocumentApprovalProcess {

	public static final String PROCESS_DEFINITION_KEY = "DocumentApproval";
```

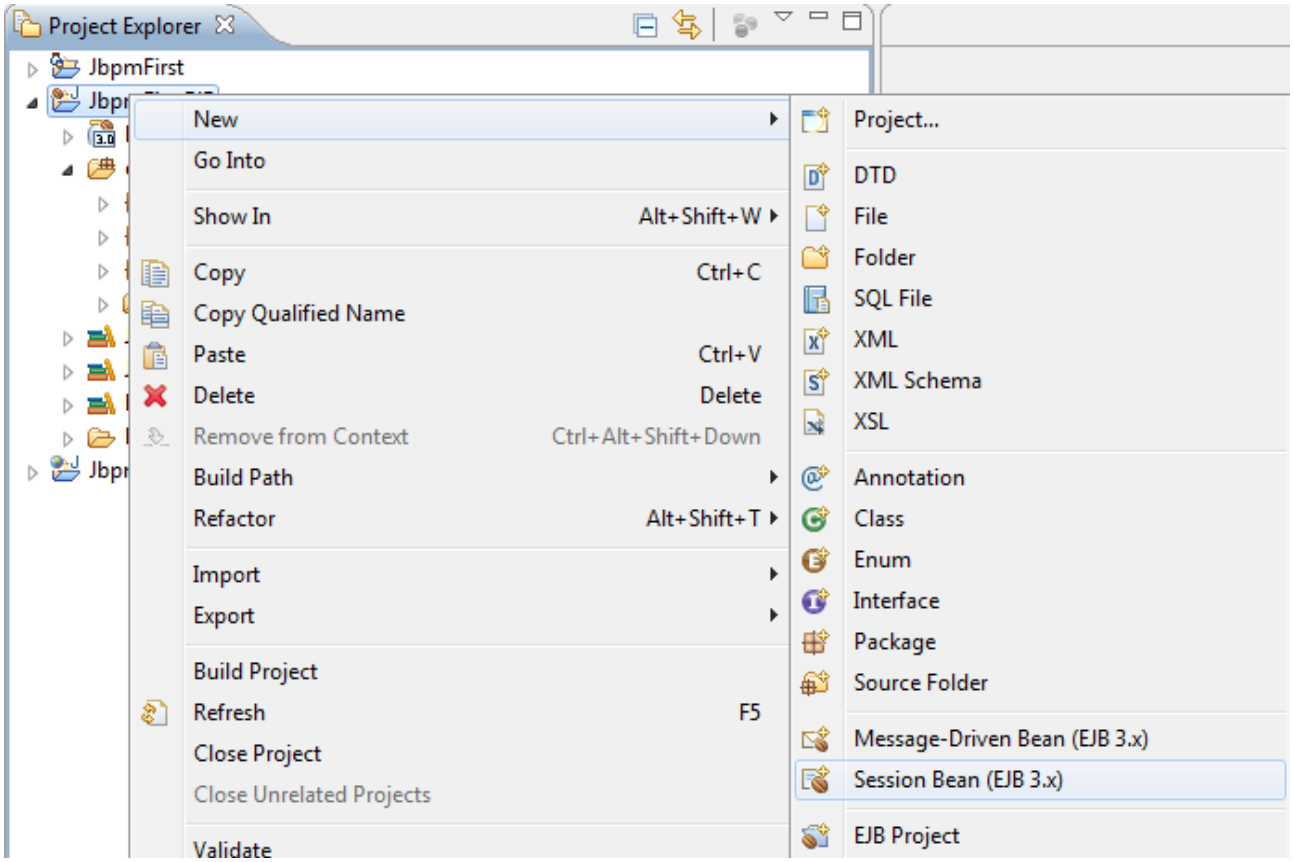
```
	private DocumentApprovalProcess() {
	}

	public static boolean start(String businessKey) {
		if (businessKey == null || businessKey.trim().equals(""))
			throw new JbpmProcessException("Invalid process instance
identifier: " + businessKey);

		// search for an already existent process instance
		if (DocumentApprovalProcess.getProcessInstance(businessKey) != null)
{
			throw new JbpmProcessException("Another process instance with
the same identifier exists: " + businessKey);
		}

		// create new process instance
		ProcessInstance processInstance =
ProcessEngineProxy.getExecutionService().startProcessInstanceByKey(DocumentAppro
valProcess.PROCESS_DEFINITION_KEY, businessKey);

		return processInstance == null ? false : true;
	}

	public static void signal(String executionId) {
		if (executionId == null || executionId.trim().equals(""))
			throw new JbpmProcessException("Invalid execution identifier:
" + executionId);

		// signal execution

	ProcessEngineProxy.getExecutionService().signalExecutionById(executionId);
	}

	public static void signal(String executionId, String signalName) {
		if (executionId == null || executionId.trim().equals(""))
			throw new JbpmProcessException("Invalid execution identifier:
" + executionId);

		// signal execution

	ProcessEngineProxy.getExecutionService().signalExecutionById(executionId,
signalName);
	}

	public static void end(String businessKey) {
		if (businessKey == null || businessKey.trim().equals(""))
			throw new JbpmProcessException("Invalid process instance
identifier: " + businessKey);

		// search for an already existent process instance
		ProcessInstance processInstance =
DocumentApprovalProcess.getProcessInstance(businessKey);
		if (processInstance == null) {
			throw new JbpmProcessException("No process instance found for
identifier: " + businessKey);
		}


ProcessEngineProxy.getExecutionService().endProcessInstance(processInstance.getI
d(), Execution.STATE_ENDED);
	}
```

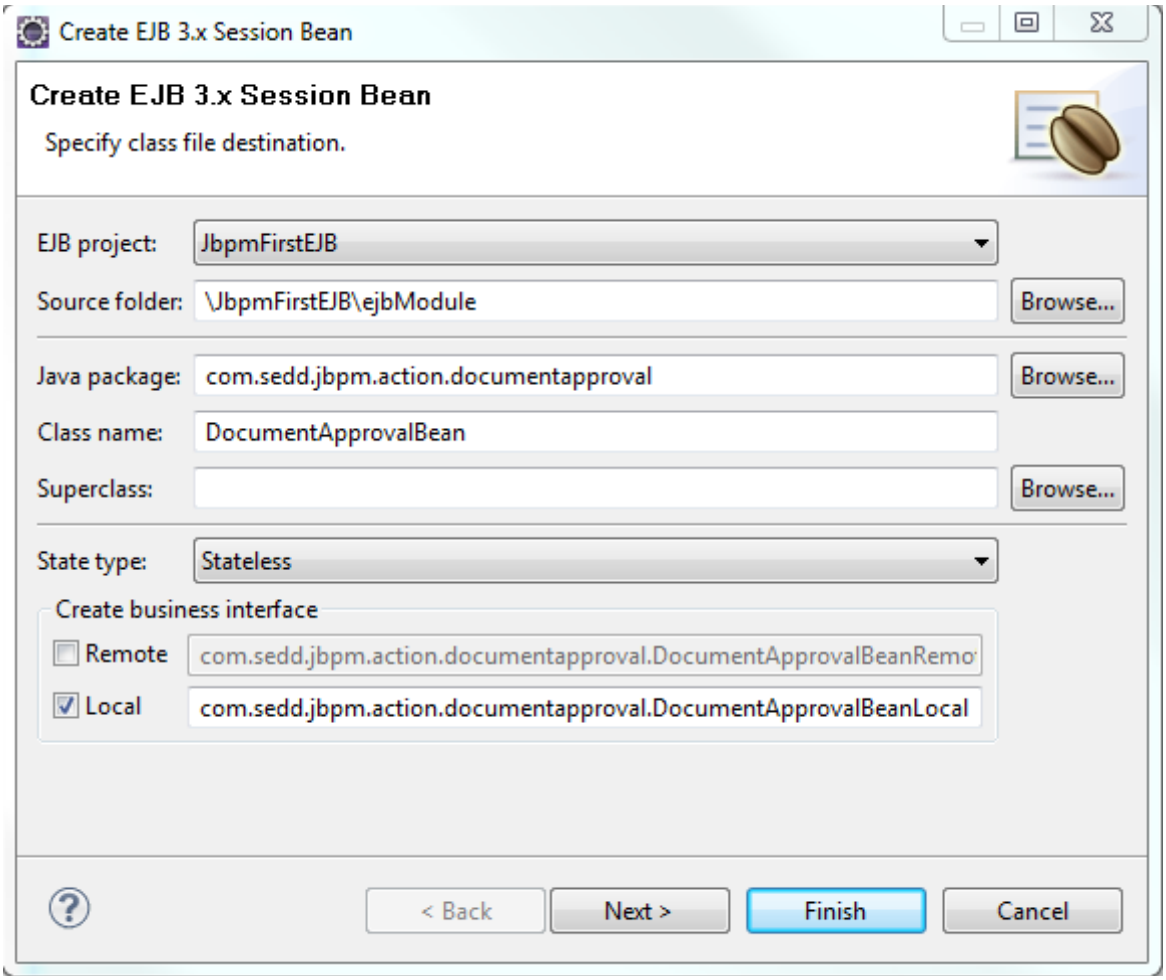
```
	private static String createProcessInstanceId(String businessKey) {
		return DocumentApprovalProcess.PROCESS_DEFINITION_KEY + "." +
businessKey;
	}

	private static ProcessInstance getProcessInstance(String businessKey) {
		return
ProcessEngineProxy.getExecutionService().findProcessInstanceById(DocumentApprova
lProcess.createProcessInstanceId(businessKey));
	}

}
```

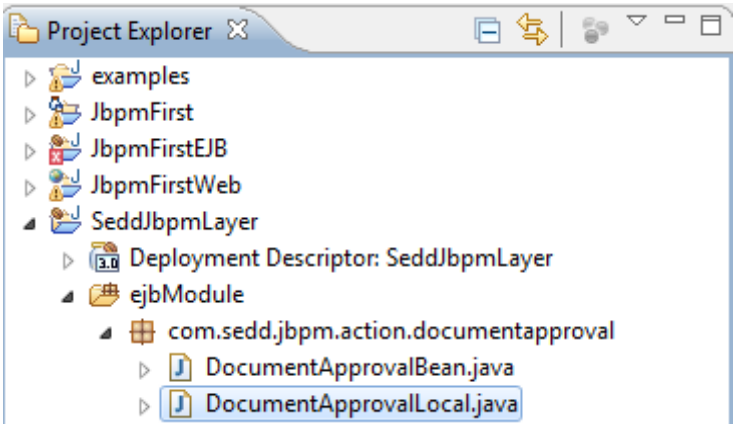
Now let's discuss a little about the code. There are two very important concepts used here: the business process definition key and the the business key. The business process definition key(process key) refers to an identifier that points to the process definition we will use. The business key refers to an identifier that points to the process instance(execution) we will use. The idea behind JBPM is that for a process definition deployed you can have many process instances running(just like the relationship between classes and objects). Long story short, the two identifiers are needed to identify the exact process instance we will be using from our business layer and the only thing that makes sense to the business layer is the business key, which it defines. The process key and the business key combine together in ways defined by the JBPM user guide, so take a look there to see how you can save/retrieve process instances.

For our specific process we need the following actions to be performed: start, signal(both simple and with tag) and end. Thus the public methods you see in this class are used by the EJB layer, which will never know what's behind the scene (it doesn't know anything about the interaction with the process engine) :-P

The next thing we will add is the session bean that implements the business logic of the application specific for this process. Because this session bean will act as an action handler for the web layer, we will add under the generic action package, in a dedicated package for clean separation.


Project Explorer
JbpmFirst
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import
Export
Build Project
Refresh F5
Close Project
Close Unrelated Projects
Validate
Project...
DTD
File
Folder
SQL File
XML
XML Schema
XSL
Annotation
Class
Enum
Interface
Package
Source Folder
Message-Driven Bean (EJB 3.x)
Session Bean (EJB 3.x)
EJB Project


Create EJB 3.x Session Bean
Create EJB 3.x Session Bean
Specify class file destination.
EJB project: JbpmFirstEJB
Source folder: \JbpmFirstEJB\ejbModule
Browse...
Java package: com.sedd.jbpm.action.documentapproval
Browse...
Class name: DocumentApprovalBean
Superclass:
Browse...
State type: Stateless
Create business interface
Remote com.sedd.jbpm.action.documentapproval.DocumentApprovalBeanRemo
Local com.sedd.jbpm.action.documentapproval.DocumentApprovalBeanLocal
< Back
Next >
Finish
Cancel



I will show you the code inside the session bean and local interface now, bu we will build the dependencies and discuss it in detail afterward.

```
package com.sedd.jbpm.action.documentapproval;

import javax.ejb.Local;

@Local
public interface DocumentApprovalBeanLocal {
	public String startNewDocumentApprovalProcess();

	public void documentSubmitted(String businessKey);

	public void documentAccepted(String businessKey);

	public void documentRejected(String businessKey);

	public void endDocumentApprovalProcess(String businessKey);

	public void notifyWaitingForCheck(String executionId, String businessKey);

	public void notifyWaitingForSubmit(String executionId, String
businessKey);

	public void notifySuccess(String executionId, String businessKey);

	public void notifyFailure(String executionId, String businessKey);
}
```

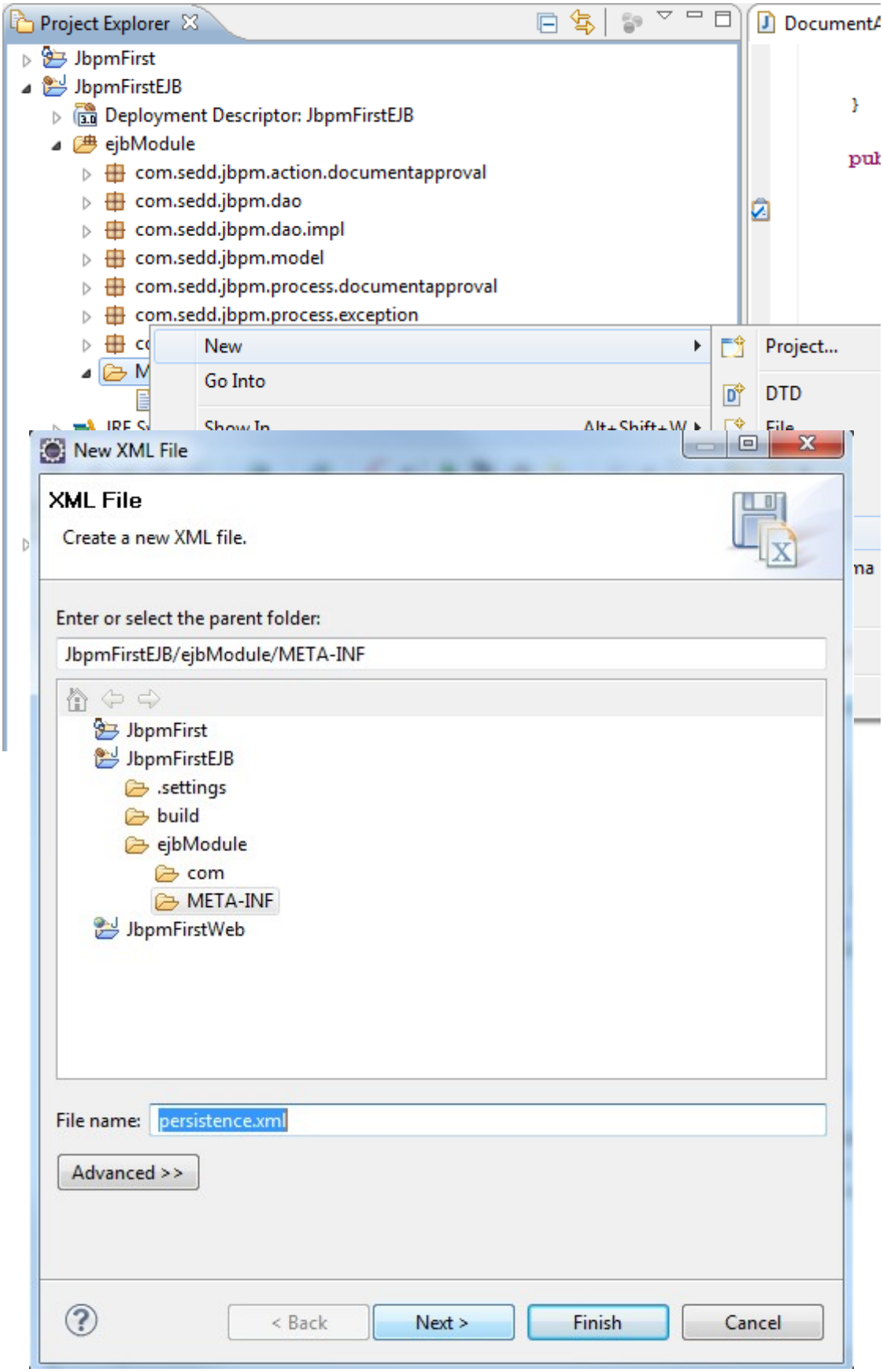
```
package com.sedd.jbpm.action.documentapproval;

import java.util.Random;

import javax.ejb.EJB;
import javax.ejb.Stateless;

import com.sedd.jbpm.dao.MemberDAO;
import com.sedd.jbpm.model.Member;
import com.sedd.jbpm.process.documentapproval.DocumentApprovalProcess;

/**
 * Session Bean implementation class DocumentApprovalBean
 */
@Stateless
```

```
public class DocumentApprovalBean implements DocumentApprovalBeanLocal {

	@EJB
	private MemberDAO memberDAO;

	/**
	 * Default constructor.
	 */
	public DocumentApprovalBean() {
		// TODO Auto-generated constructor stub
	}

	public String startNewDocumentApprovalProcess() {
		System.out.println("public startNewDocumentApprovalProcess()
{...}");

		// update database
		Member member = this.memberDAO.find(0);
		member.setTagline("public startNewDocumentApprovalProcess() {...}");
		this.memberDAO.persist(member);

		// start process
		String businessKey = Long.toString(new Random().nextLong());
		boolean success = DocumentApprovalProcess.start(businessKey);
		return success ? businessKey : null;
	}

	public void documentSubmitted(String businessKey) {
		System.out.println("public documentSubmitted() {...}");

		// TODO: get execution id from database
		Member member = this.memberDAO.find(0);
		String executionId = member.getTagline();

		DocumentApprovalProcess.signal(executionId);
	}

	public void documentAccepted(String businessKey) {
		System.out.println("public documentAccepted() {...}");
		this.documentChecked(businessKey, "accept");
	}

	public void documentRejected(String businessKey) {
		System.out.println("public documentRejected() {...}");
		this.documentChecked(businessKey, "reject");
	}

	private void documentChecked(String businessKey, String signalName) {
		System.out.println("public documentChecked() {...}");

		// TODO: get execution id from database
		Member member = this.memberDAO.find(0);
		String executionId = member.getTagline();

		DocumentApprovalProcess.signal(executionId, signalName);
	}

	public void endDocumentApprovalProcess(String businessKey) {
		System.out.println("public endDocumentApprovalProcess() {...}");

		// update database
		Member member = this.memberDAO.find(0);
		member.setTagline("public endDocumentApprovalProcess() {...}");
		this.memberDAO.persist(member);
```

```
            // end process
            DocumentApprovalProcess.end(businessKey);
      }

      public void notifyWaitingForCheck(String executionId, String businessKey)
{
            System.out.println("public notifyWaitingForCheck() {...}");
            // TODO: do some work on the application database and store
execution id

            Member member = this.memberDAO.find(0);
            member.setTagline(executionId);
            this.memberDAO.persist(member);
      }

      public void notifyWaitingForSubmit(String executionId, String businessKey)
{
            System.out.println("public notifyWaitingForSubmit() {...}");
            // TODO: do some work on the application database and store
execution id

            Member member = this.memberDAO.find(0);
            member.setTagline(executionId);
            this.memberDAO.persist(member);
      }

      public void notifySuccess(String executionId, String businessKey) {
            System.out.println("public notifySuccess() {...}");
            // TODO: do some work on the application database and store
execution id

            Member member = this.memberDAO.find(0);
            member.setTagline(executionId);
            this.memberDAO.persist(member);
      }

      public void notifyFailure(String executionId, String businessKey) {
            System.out.println("public notifyFailure() {...}");
            // TODO: do some work on the application database and store
execution id

            Member member = this.memberDAO.find(0);
            member.setTagline(executionId);
            this.memberDAO.persist(member);
      }

}
```

The first dependency is MemberDAO, used for data persistence. I will not go into much details since the purpose of this tutorial is JBPM and not JPA. For those unfamiliar with java persistence, this is a way of abstracting the communication with the database.

So, it's now time to see how we can configure the database communication for our specific application. First of all, we will use JPA with Hibernate and here's what we need to do:

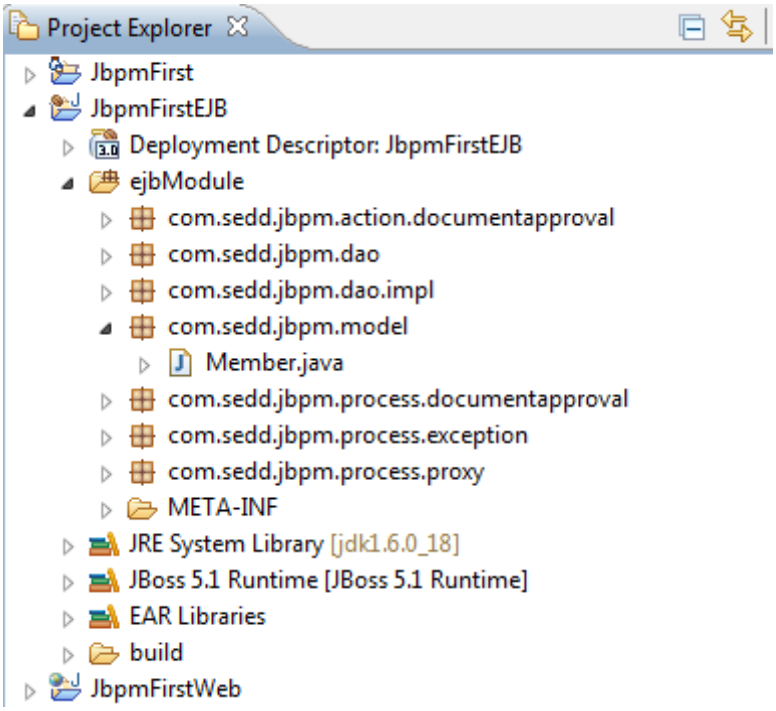


Place the following content in that file

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Persistence deployment descriptor for dev profile -->
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
```

```
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
            version="1.0">

   <persistence-unit name="SeddJbpm">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/SeddJbpmDS</jta-data-source>
      <!-- The <jar-file> element is necessary if you put the persistence.xml in
the WAR and the classes in the JAR -->
      <!--
      <jar-file>../../vehicles.jar</jar-file>
      -->
      <properties>
         <property name="hibernate.dialect"
value="org.hibernate.dialect.Oracle10gDialect"/>
         <property name="hibernate.hbm2ddl.auto" value="update"/>
         <property name="hibernate.show_sql" value="true"/>
         <property name="hibernate.format_sql" value="true"/>
         <property name="jboss.entity.manager.factory.jndi.name"
value="java:/SeddJbpmEntityManagerFactory"/>
      </properties>
   </persistence-unit>

</persistence>
```

That's all you need to do for the database communication. Notice that we are using the data source we have defined in the EAR and deployed to JBoss. Now we need to define the database schema, but for that we rely on Hibernate to generate it out of the model classes we will create – enjoying the goods of a great ORM tool.
For this example we will use a single class called Member which I admit copying from a JBoss Seam example. For this application we will use a single database schema thus we should create all our model classes under the model package.




```
package com.sedd.jbpm.model;
```

```java
import java.io.Serializable;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
import javax.persistence.UniqueConstraint;

import org.hibernate.validator.Email;
import org.hibernate.validator.Length;
import org.hibernate.validator.NotNull;
import org.hibernate.validator.Pattern;

@Entity
@Table(uniqueConstraints = @UniqueConstraint(columnNames = "membername"))
public class Member implements Serializable {

	private static final long serialVersionUID = 361851279105935874L;

	public enum Gender {
		male("Male", "his"), female("Female", "her");

		private String descr;
		private String possessive;

		Gender(String descr, String possessive) {
			this.descr = descr;
			this.possessive = possessive;
		}

		public String getDescr() {
			return descr;
		}

		public String getPossessive() {
			return possessive;
		}
	};

	private Integer memberId;
	private String memberName;
	private String firstName;
	private String lastName;
	private String email;

	private String tagline;
	private Gender gender;
	private Date dob;
	private String location;
	private Date memberSince;

	@Id
	@GeneratedValue
	public Integer getMemberId() {
		return memberId;
	}

	public void setMemberId(Integer memberId) {
		this.memberId = memberId;
	}
```

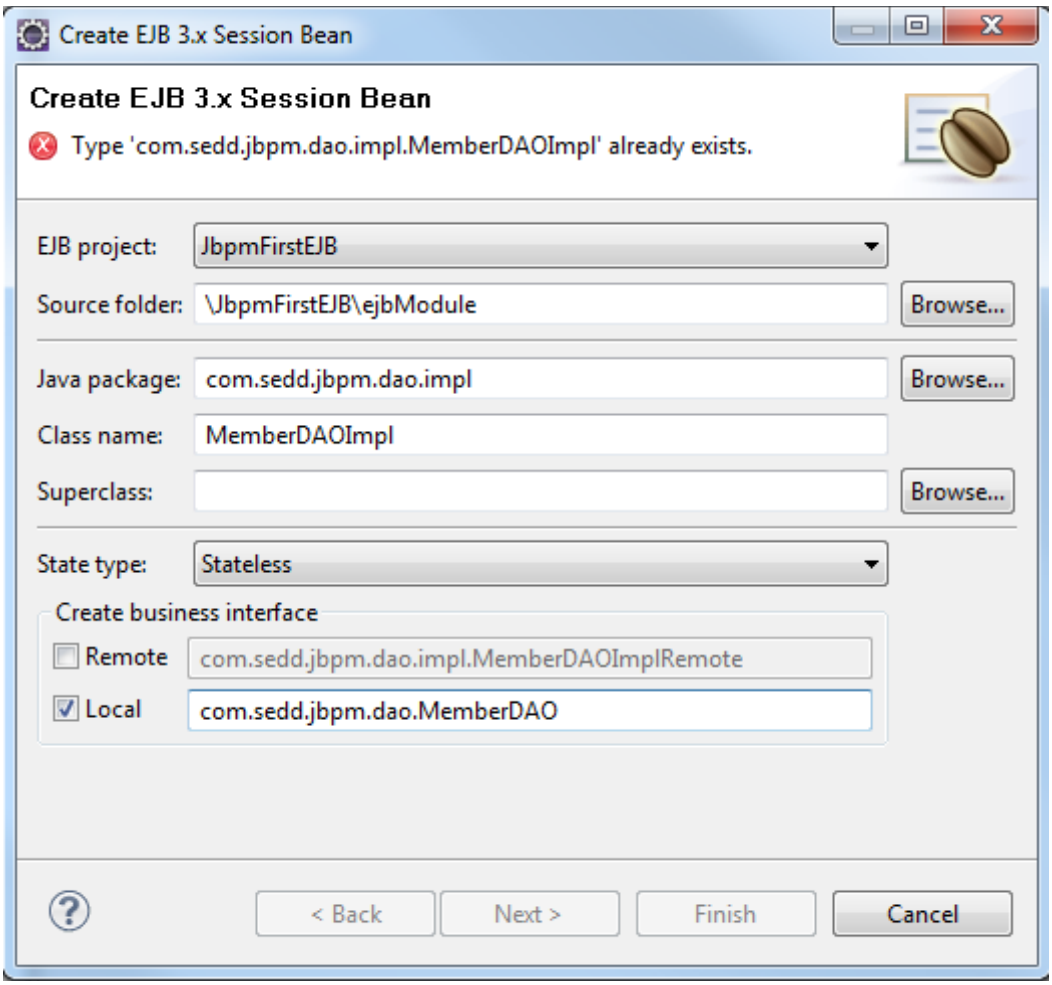
```
	@NotNull
	@Length(min = 3, max = 40)
	@Pattern(regex = "[a-zA-Z]?[a-zA-Z0-9_]+", message = "Member name must
start with a letter, and only contain letters, numbers or underscores")
	public String getMemberName() {
		return memberName;
	}

	public void setMemberName(String memberName) {
		this.memberName = memberName;
	}

	@NotNull
	@Length(min = 3, max = 40)
	@Pattern(regex = "[a-zA-Z]+", message = "First name must only contain
letters")
	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	@NotNull
	@Length(min = 3, max = 40)
	@Pattern(regex = "[a-zA-Z]+", message = "Last name must only contain
letters")
	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	@NotNull
	@Email
	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	@NotNull
	public Date getDob() {
		return dob;
	}

	public void setDob(Date dob) {
		this.dob = dob;
	}

	@NotNull
	public Gender getGender() {
		return gender;
	}

	public void setGender(Gender gender) {
		this.gender = gender;
	}
```

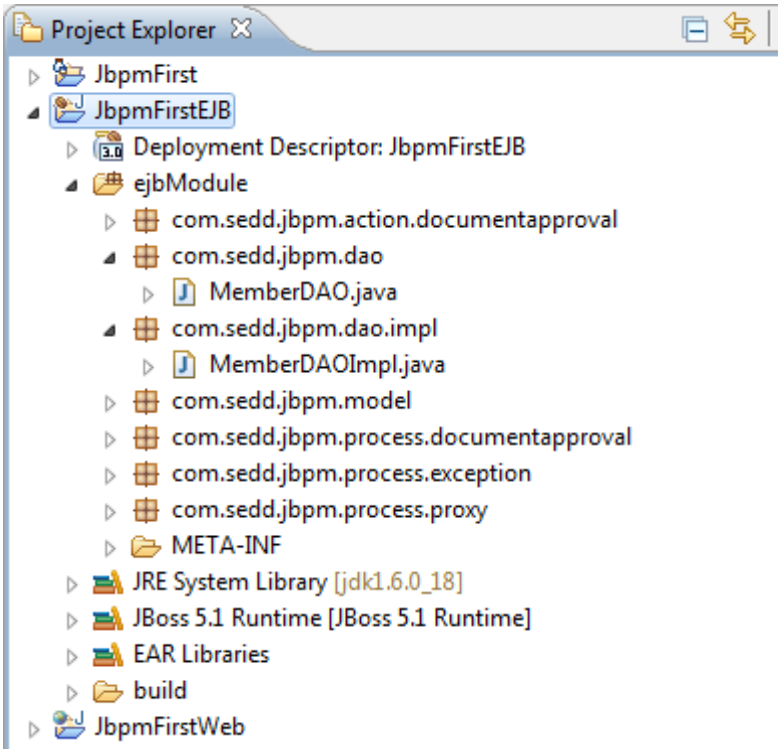
```
	public String getLocation() {
		return location;
	}

	public void setLocation(String location) {
		this.location = location;
	}

	@NotNull
	public Date getMemberSince() {
		return memberSince;
	}

	public void setMemberSince(Date memberSince) {
		this.memberSince = memberSince;
	}

	public String getTagline() {
		return tagline;
	}

	public void setTagline(String tagline) {
		this.tagline = tagline;
	}

	@Transient
	public String getAge() {
		Calendar birthday = new GregorianCalendar();
		birthday.setTime(dob);
		int by = birthday.get(Calendar.YEAR);
		int bm = birthday.get(Calendar.MONTH);
		int bd = birthday.get(Calendar.DATE);

		Calendar now = new GregorianCalendar();
		now.setTimeInMillis(System.currentTimeMillis());
		int ny = now.get(Calendar.YEAR);
		int nm = now.get(Calendar.MONTH);
		int nd = now.get(Calendar.DATE);

		int age = ny - by + (nm > bm || (nm == bm && nd >= bd) ? 0 : -1);
		return String.format("%d years old", age);
	}
}
```

Again, I will skip the JPA/Hiebrnate specific details. The main idea is that we will use the Member class to represent a row in a database table and rely on the used tools to figure out the rest of the work: how to manipulate the database schema and data and how to communicate with the database.

The next thing we want to do is pus some sort of abstraction layer between our application code and the JPA tools and that's what we are doing with the DAOs. You can find more about the benefits of using the DAO design pattern on web.

Since we are in the EJB world, we'll use a stateless session bean(SLSB) to implement this DAO.


Create EJB 3.x Session Bean
Create EJB 3.x Session Bean
Type 'com.sedd.jbpm.dao.impl.MemberDAOImpl' already exists.
EJB project:
JbpmFirstEJB
Source folder:
\JbpmFirstEJB\ejbModule
Browse...
Java package:
com.sedd.jbpm.dao.impl
Browse...
Class name:
MemberDAOImpl
Superclass:
Browse...
State type:
Stateless
Create business interface
Remote
com.sedd.jbpm.dao.impl.MemberDAOImplRemote
Local
com.sedd.jbpm.dao.MemberDAO
< Back
Next >
Finish
Cancel

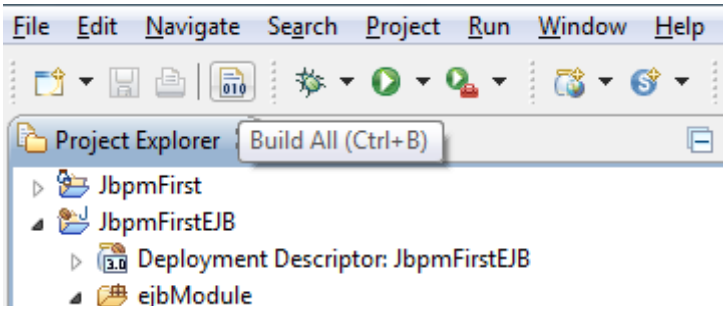

Project Explorer
JbpmFirst
JbpmFirstEJB
Deployment Descriptor: JbpmFirstEJB
ejbModule
com.sedd.jbpm.action.documentapproval
com.sedd.jbpm.dao
MemberDAO.java
com.sedd.jbpm.dao.impl
MemberDAOImpl.java
com.sedd.jbpm.model
com.sedd.jbpm.process.documentapproval
com.sedd.jbpm.process.exception
com.sedd.jbpm.process.proxy
META-INF
JRE System Library [jdk1.6.0_18]
JBoss 5.1 Runtime [JBoss 5.1 Runtime]
EAR Libraries
build
JbpmFirstWeb

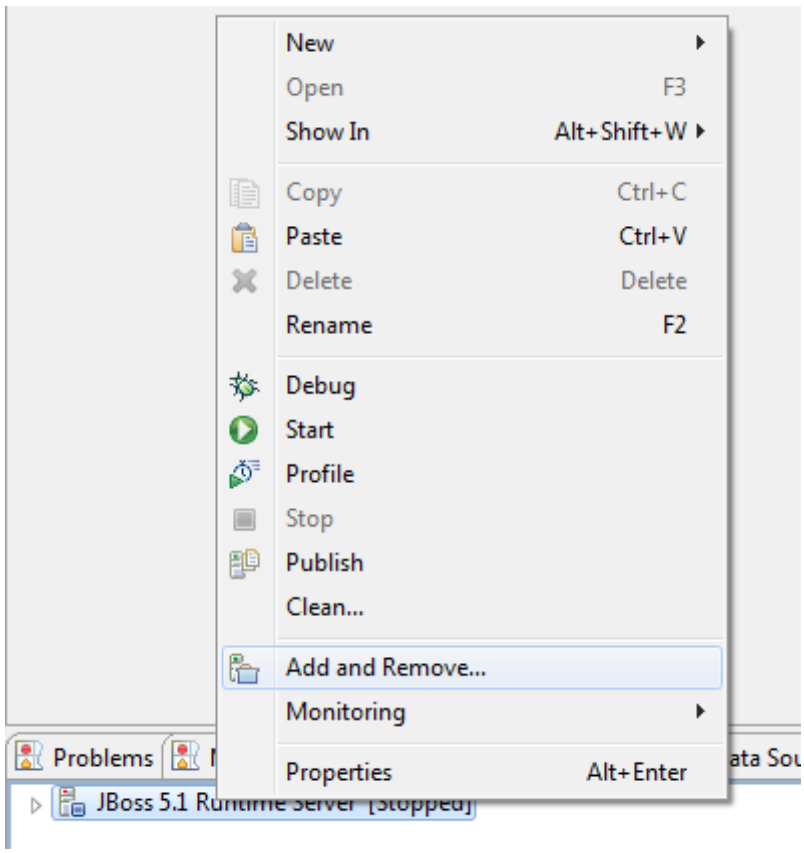
And the code:

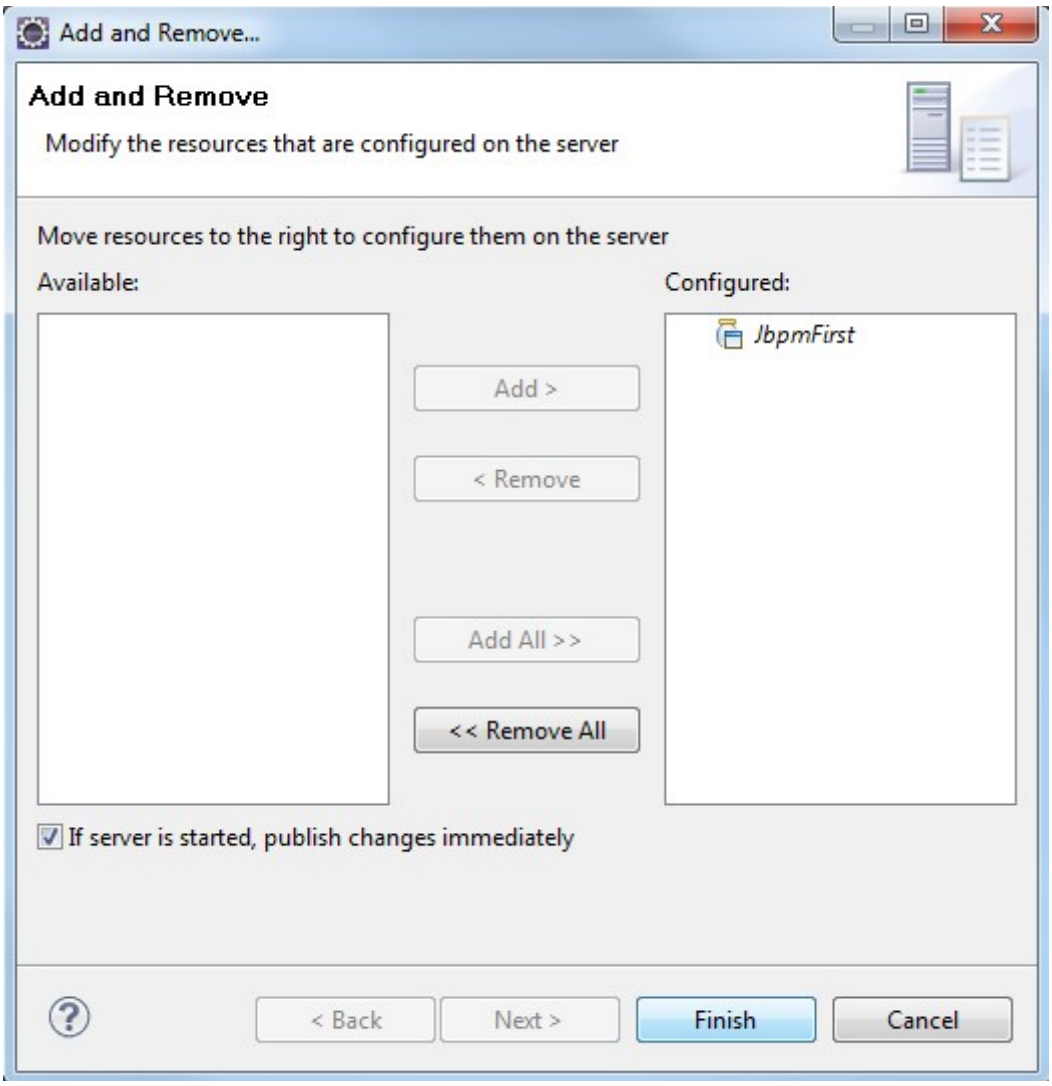
```java
package com.sedd.jbpm.dao;

import javax.ejb.Local;

import com.sedd.jbpm.model.Member;

@Local
public interface MemberDAO {
	public Member find(Integer memberId);

	public void persist(Member member);
}
```

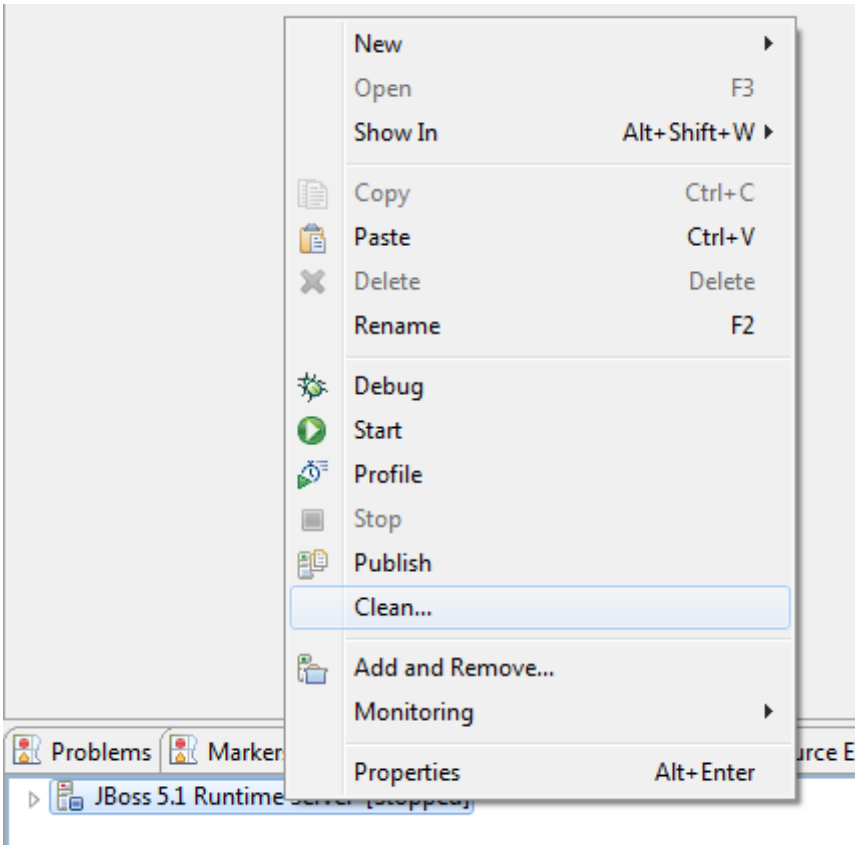
```java
package com.sedd.jbpm.dao.impl;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.sedd.jbpm.dao.MemberDAO;
import com.sedd.jbpm.model.Member;

@Stateless
public class MemberDAOImpl implements MemberDAO {

	@PersistenceContext(unitName = "SeddJbpm")
	private EntityManager entityManager;

	public Member find(Integer memberId) {
		return this.entityManager.find(Member.class, memberId);
	}

	public void persist(Member member) {
		this.entityManager.persist(member);
	}

}
```

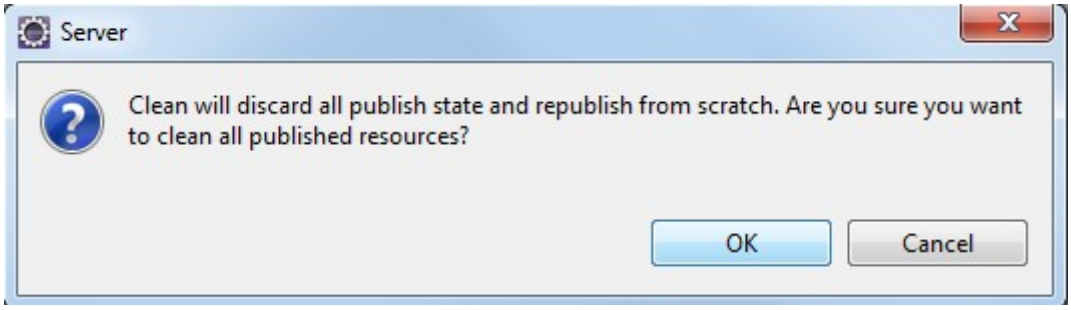
Notice how we use the persistence context we defined in persistence.xml. Let's test what we have so far to see if we're OK. Build the project and publish it to JBoss.




New
Open F3
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Rename F2
Debug
Start
Profile
Stop
Publish
Clean...
Add and Remove...
Monitoring
Properties Alt+Enter
Problems
JBoss 5.1 Runtime Server [Stopped]


Add and Remove...
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
JbpmFirst
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Finish
Cancel





You should see no error message and there should be JNDI bindings for MemberDAO and JPA mappings for Member class.

```
with dependencies:
and demands:
   jboss.ejb:service=EJBTimerService
   jndi:JbpmFirst/MemberDAOImpl/local-com.sedd.jbpm.dao.MemberDAO
```

```
[Ejb3Configuration] Persistence provider caller does not implement the EJB3 spec cc
[AnnotationBinder] Binding entity from annotated class: com.sedd.jbpm.model.Member
[EntityBinder] Bind entity com.sedd.jbpm.model.Member on table Member
```

So, now that we know how we interact with our database, it's time to go back to our DocumentApprovalBean SLSB. We have the following business logic to implement(remember that this is purely for demonstration purpose, in a real life situation a few things may be missing or actually make sense, but this is not the case ;-)): the user will press a button to start the process; then he will have to approve or reject a request for document submission; in case of acceptance, the user is prompted for document submission and then the process will end successfully; in case of rejection, the process will end with an error; the user also has the possibility to stop the process at

any time.
Let's take the start process method:

```
public String startNewDocumentApprovalProcess() {
    System.out.println("public startNewDocumentApprovalProcess() {...}");

    // update database
    Member member = this.memberDAO.find(0);
    member.setTagline("public startNewDocumentApprovalProcess() {...}");
    this.memberDAO.persist(member);

    // start process
    String businessKey = Long.toString(new Random().nextLong());
    boolean success = DocumentApprovalProcess.start(businessKey);
    return success ? businessKey : null;
}
```

We want to display messages every time something is happening so don't mind the System.out lines. Then we want to do a database change to see how the CMT JTA behaves on the configuration we have. For this we will use the member.tagline property. Just store whatever in there, it doesn't have to make sense, as long as it helps you identify the state of the application.

NOTE: manually insert a record in the member table in the sedd database, with id set to 0.

Next we need a business key, in real life this business key could be an order document number or a shipment id, etc. Basically you need an information which is unique in your database context, here we just generate a random number. Notice how we use the abstraction layer to start the process; the SLSB knows absolutely nothing about the process, only that it should use that specific business key if he wants to use that specific process again. This way, if anything changes in the JBPM API, we only have to change the DocumentApprovalProcess class.

NOTE: you probably must have noticed that we didn't use session beans, nor even class instances when defining the JBPM abstraction layer. We can use static methods because the JBPM ProcessEngine is thread safe and because we use SLSB to interact with the abstraction layer thus we do not and shouldn't store any state in the abstraction layer. So, using static methods forces us to make the design in such a way that we'll avoid holding any state in the abstraction layer.

If everything is OK, the start method returns the business key used to start and identify the process to the caller. All other calls should be parametrized with this business key, in order to affect the same process.

After start up, the process enters a waiting state, where the user must accept or reject. For this purpose we have the following methods:

```
public void documentAccepted(String businessKey) {
    System.out.println("public documentAccepted() {...}");
    this.documentChecked(businessKey, "accept");
}

public void documentRejected(String businessKey) {
    System.out.println("public documentRejected() {...}");
    this.documentChecked(businessKey, "reject");
}

private void documentChecked(String businessKey, String signalName) {
    System.out.println("public documentChecked() {...}");

    // TODO: get execution id from database
    Member member = this.memberDAO.find(0);
    String executionId = member.getTagline();

    DocumentApprovalProcess.signal(executionId, signalName);
}
```

This is how we signal the process to continue to the next state taking the transition determined by the signal name we send to it. Every interaction with the process happens in a similar manner, feel free to study all other methods, except for those that have an executionId parameter(they are a different story).

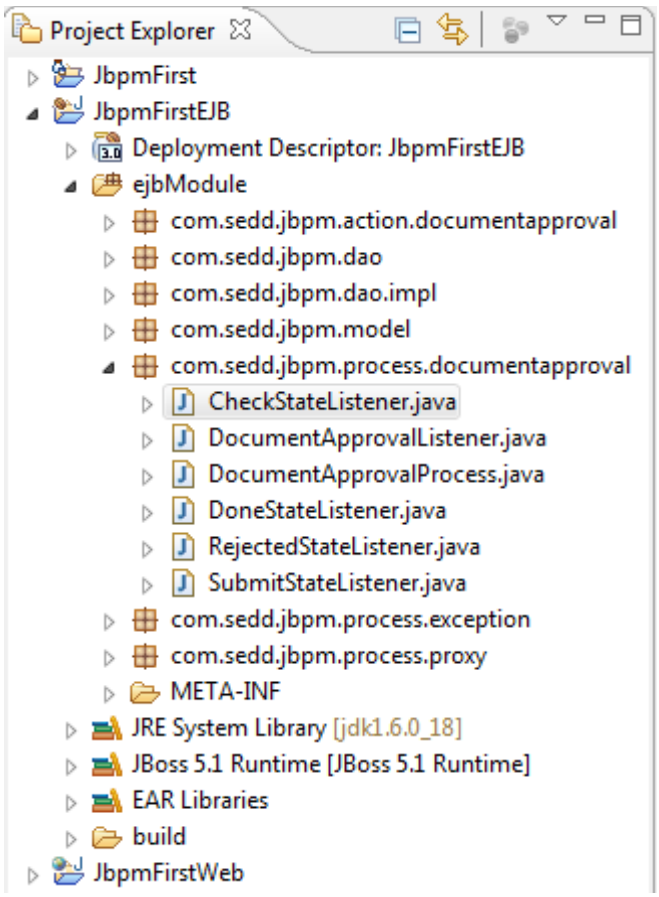
COGNITIVE: I believe you noticed that the application logic follows the path of the process and it isn't determined by the path of the process. In other words: we built the application with a certain degree of knowledge about the current state of the process and what should happen next. I'm new to JBPM so I'm struggling between trying to write an application which is 100% determined by the process(still I'm not finding enough methods in the APIs for this) and trying to design an application that is dependent on the business process, without duplicating the logic described in it. Thus you may find my approach totally opposed to the purpose of JBPM and in this case I will appreciate if you point me to some good examples :-). Thanks.

It's now time to discuss about the 'special' methods in our SLSB, the executionId ones. If you read the JBPM user guide you will notice an interesting indication: it's a wise decision to use event listeners on state changes instead of 'remembering' the next state of the process. I do like to be wise and I'm following the indication. This is the process definition decorated with some event listeners that will help me notify the business layer about the current state of the process.

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="DocumentApproval" xmlns="http://jbpm.org/4.3/jpdl">

    <start name="start">
        <transition to="check"/>
    </start>

    <state name="check">
        <on event="start">
            <event-listener
```

```
class="com.sedd.jbpm.process.documentapproval.CheckStateListener" />
                </on>
                <transition name="reject" to="rejected"/>
                <transition name="accept" to="submit"/>
        </state>

        <state name="submit">
                <on event="start">
                        <event-listener
class="com.sedd.jbpm.process.documentapproval.SubmitStateListener" />
                </on>
                <transition to="done"/>
        </state>

        <end name="done">
                <on event="start">
                        <event-listener
class="com.sedd.jbpm.process.documentapproval.DoneStateListener" />
                </on>
        </end>

        <end-error name="rejected">
                <on event="start">
                        <event-listener
class="com.sedd.jbpm.process.documentapproval.RejectedStateListener" />
                </on>
        </end-error>

</process>
```

Because I'm trying to keep everything related to JBPM in the abstraction layer and also group everything according to a business orientation, I'm creating all listeners in the com.sedd.jbpm.process.documentapproval package. Since all listeners are similar, I will only talk about one of them.




```
package com.sedd.jbpm.process.documentapproval;

import org.jbpm.api.listener.EventListener;
import org.jbpm.api.listener.EventListenerExecution;

public class CheckStateListener extends DocumentApprovalListener implements
EventListener {

	private static final long serialVersionUID = 2613244678350181265L;

	public void notify(EventListenerExecution execution) throws Exception {
		System.out.println("public void notify(EventListenerExecution
execution) throws Exception {...}");

	this.getDocumentApprovalBean().notifyWaitingForCheck(execution.getId(),
execution.getKey());
	}

}
```

All this event listener does is notify the SLSB about the current state of the process. You can see that it sends the execution id as a parameter so that the SLSB know which is the execution that triggered the event(the business key - `execution.getKey()` - is not always enough because it only identifies the main execution path of the process while a process can have many concurrent paths of execution). This is one place where our application is no longer 100% decoupled from the JBPM details. The SLSB will store the executionId in the database to be able to recognize the current execution next time it will be invoked. This scenario can only be avoided if there is only one path of execution in a process or if the states for concurrent executions don't have to wait for external triggers.

Enough with the chatty since this is not the most interesting break from the 'clean' separation of layers. As you can see our event listener extends a base event listener, called DocumentApprovalListener. This guy's purpose is to provide a DocumentApprovalBean reference to the specific listener. Let's see the class:

```
package com.sedd.jbpm.process.documentapproval;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import com.sedd.jbpm.action.documentapproval.DocumentApprovalBeanLocal;
import com.sedd.jbpm.process.exception.JbpmProcessException;

public abstract class DocumentApprovalListener {

	private static final String JNDI_NAME =
"JbpmFirst/DocumentApprovalBean/local";

	private DocumentApprovalBeanLocal documentApprovalBean;

	protected DocumentApprovalBeanLocal getDocumentApprovalBean() {
		if (this.documentApprovalBean == null) {
			try {
				InitialContext ctx = new InitialContext();
				this.documentApprovalBean = (DocumentApprovalBeanLocal)
ctx.lookup(JNDI_NAME);
			} catch (NamingException e) {
				throw new JbpmProcessException("Could not find document
approval bean.", e);
			}
		}
		return this.documentApprovalBean;
	}

}
```

The only thing it does is look up our SLSB in the JVM context. Why is this necessary and why don't we do the database update in the event listener itself is the subject of the following discussion.

Let's start with the former: why don't we do the database update in the listener? Well, it's simple, because the listener is only aware of what the JBPM is aware, thus the jbpm database and not the sedd database. If we had our application using the same database as the JBPM engine, then we could have used the listener to make database changes. But this would have broken a separation of concerns principle: let's do the the business changes in in the application layer and the JBPM changes in the JBPM layer. So we're stuck with notifying the SLSB about the changes occurred in the process and let the SLSB react on it's own on those changes.
So now let's try to answer the first question: why can't we make the listener a SLSB and let the EJB container inject the DocumentApprovalBean? This answer is not obvious and I have to admit that I tested it first to see if it works, before I implemented things this way. It doesn't work if you make a SLSB out of the listener! One plausible answer(not confirmed yet) is this: the JBPM process engine instantiates the listener by Java Reflection and uses that instance which is not managed by the EJB container, thus we loose all container related facilities and we cannot rely on injection. If you try it, you will notice a NPE exception when you'll try to use the injected bean. Have fun with it :-) !

I don't like this solution but till someone else comes with a better one or a suggestion for another approach, we're stuck with this coding style. Enjoy!

So now you know who and how it's calling this method:

```
public void notifyWaitingForCheck(String executionId, String businessKey) {
    System.out.println("public notifyWaitingForCheck() {...}");
    // TODO: do some work on the application database and store execution id

    Member member = this.memberDAO.find(0);
    member.setTagline(executionId);
    this.memberDAO.persist(member);
}
```

You can see that in here we store the execution id in our database to know where to continue from next time.

All other listeners do similar things. This is the code:

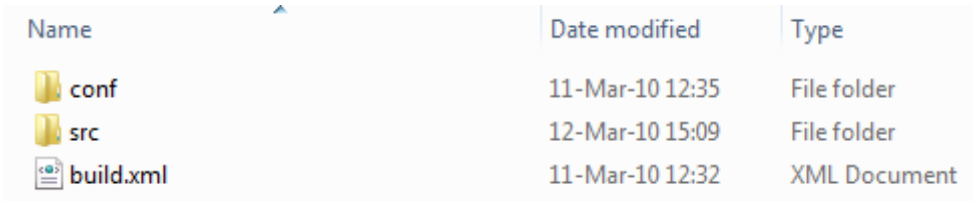
```
package com.sedd.jbpm.process.documentapproval;

import org.jbpm.api.listener.EventListener;
import org.jbpm.api.listener.EventListenerExecution;

public class DoneStateListener extends DocumentApprovalListener implements
EventListener {

	private static final long serialVersionUID = -3354162536385951836L;

	public void notify(EventListenerExecution execution) throws Exception {
		this.getDocumentApprovalBean().notifySuccess(execution.getId(),
execution.getKey());
	}

}
```

```
package com.sedd.jbpm.process.documentapproval;

import org.jbpm.api.listener.EventListener;
import org.jbpm.api.listener.EventListenerExecution;

public class RejectedStateListener extends DocumentApprovalListener implements
EventListener {

	private static final long serialVersionUID = -5979894986808100172L;

	public void notify(EventListenerExecution execution) throws Exception {
		this.getDocumentApprovalBean().notifyFailure(execution.getId(),
execution.getKey());
	}

}
```

```
package com.sedd.jbpm.process.documentapproval;

import org.jbpm.api.listener.EventListener;
import org.jbpm.api.listener.EventListenerExecution;

public class SubmitStateListener extends DocumentApprovalListener implements
EventListener {
```

```
	private static final long serialVersionUID = -6850159718782703924L;

	public void notify(EventListenerExecution execution) throws Exception {

	this.getDocumentApprovalBean().notifyWaitingForSubmit(execution.getId(),
execution.getKey());
	}

}
```

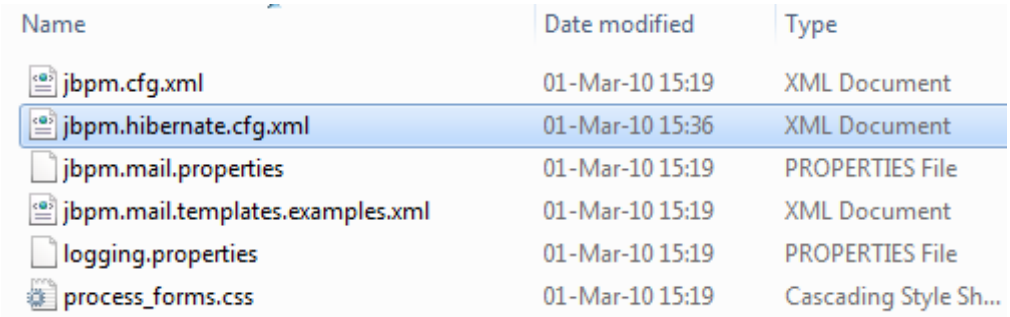
Before we can move on to the WEB layer, it's time to deploy the process to the jbpm database. We will move out of Eclipse for this. First this you have to do is select a folder, mine is D:\Viorel\jbpmtests\jbpm_process_deployment and inside I have:



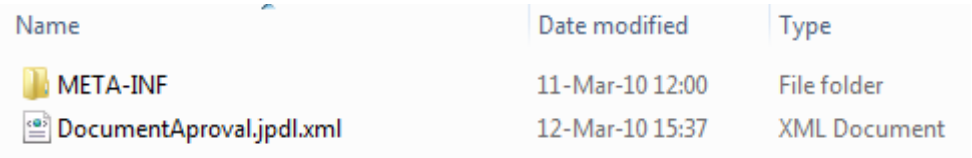
build.xml has the following content(adapted from the JBPM distribution):

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="jbpm.deployer">

	<property name="project.dir" value="./" />
	<property name="jbpm.home" value="D:/Viorel/jbpm-4.3" />

	<target name="jbpm.libs.path">
		<path id="jbpm.libs.incl.dependencies">
			<pathelement location="${project.dir}/conf" />
			<fileset dir="${jbpm.home}">
				<include name="jbpm.jar" />
			</fileset>
			<fileset dir="${jbpm.home}/lib" />
		</path>
	</target>

	<target name="deploy" depends="jbpm.libs.path">
		<jar destfile="${project.dir}/jbpm_process_deployment.bar">
			<fileset dir="${project.dir}/src" />
		</jar>
		<taskdef name="jbpm-deploy"
classname="org.jbpm.pvm.internal.ant.JbpmDeployTask"
classpathref="jbpm.libs.incl.dependencies" />
		<jbpm-deploy file="${project.dir}/jbpm_process_deployment.bar" />
	</target>

</project>
```

I'm sure that who is reading this tutorial is also able to change the properties according to his machine so I'll leave it as is. The content of the conf folder:



These files can be copied from the JBPM distribution D:\Viorel\jbpm-4.3\examples\src but make sure you edit jbpm.hibernate.cfg.xml so that it points to your jbpm database. The src folder:

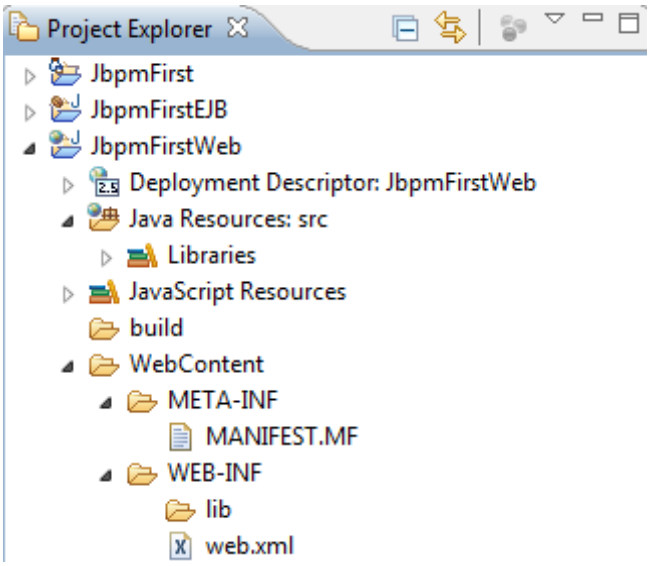


contains the process definition and the META-INF specific to every .jar: inside there's a MANIFEST.MF file with the following content:

Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 16.0-b13 (Sun Microsystems Inc.)

That's it. Now just open a command prompt and go to D:\Viorel\jbpmtests\jbpm_process_deployment where you type 'ant deploy' and you're set – deployment complete. But only if you see BUILD SUCCESSFUL ;-) .

Now let's see the WEB layer.



Edit web.xml and add JSF configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      id="WebApp_ID" version="2.5">
      <display-name>JbpmFirstWeb</display-name>
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
            <welcome-file>index.htm</welcome-file>
            <welcome-file>index.jsp</welcome-file>
            <welcome-file>default.html</welcome-file>
            <welcome-file>default.htm</welcome-file>
            <welcome-file>default.jsp</welcome-file>
      </welcome-file-list>
      <servlet>
            <servlet-name>Faces Servlet</servlet-name>
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>/faces/*</url-pattern>
      </servlet-mapping>
</web-app>
```

Next create a pages folder to put our JSF pages in. Inside, create a document_approval.jsp with the following content:

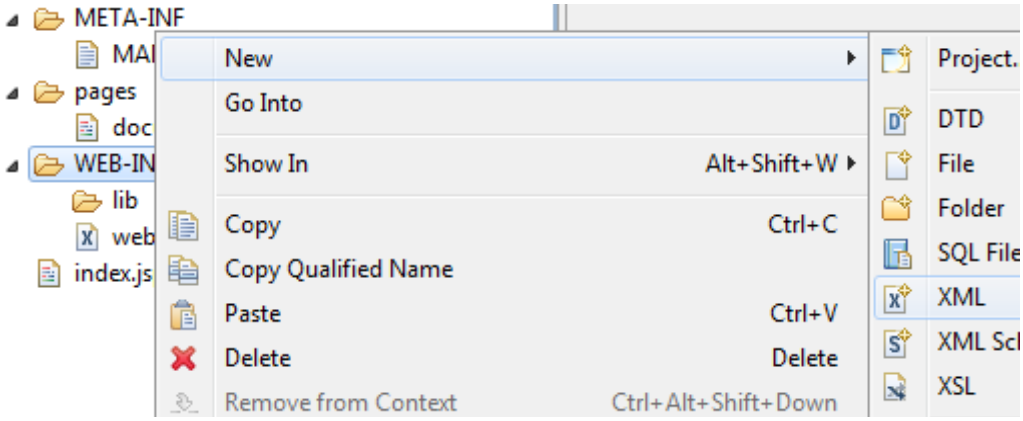
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<f:view>
      <h:form>
            <h:commandButton disabled="#{!(!documentApproval.started)}"
action="#{documentApproval.startAction}" value="Start"></h:commandButton>
            <h:commandButton disabled="#{!(documentApproval.started && !
documentApproval.checked)}" action="#{documentApproval.approveAction}"
value="Approve"></h:commandButton>
            <h:commandButton disabled="#{!(documentApproval.started && !
documentApproval.checked)}" action="#{documentApproval.rejectAction}"
value="Reject"></h:commandButton>
            <h:commandButton disabled="#{!(documentApproval.started &&
documentApproval.accepted && !documentApproval.submitted)}"
action="#{documentApproval.submitAction}" value="Submit"></h:commandButton>
            <h:commandButton disabled="#{!(documentApproval.started && !
documentApproval.rejected && !documentApproval.submitted && !
```

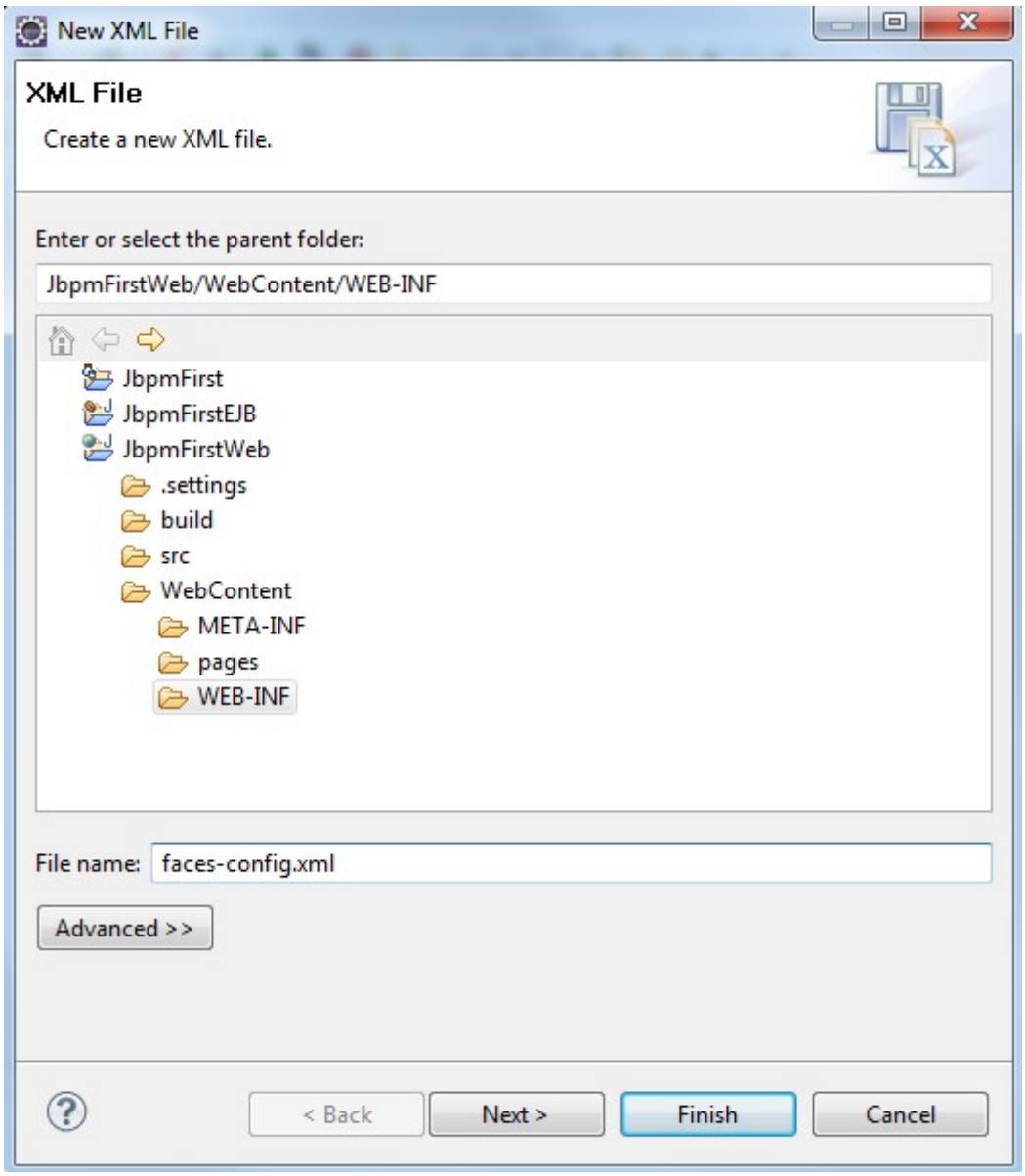
```
documentApproval.ended)}"
                    action="#{documentApproval.endAction}"
value="End"></h:commandButton>
      </h:form>
</f:view>
</body>
</html>
```

Just put a index.jsp page in the WebContent folder with the following content:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:forward page="/faces/pages/document_approval.jsp"></jsp:forward>
</body>
</html>
```

Now, the most difficult part :-) facel-config.xml and the backing bean.

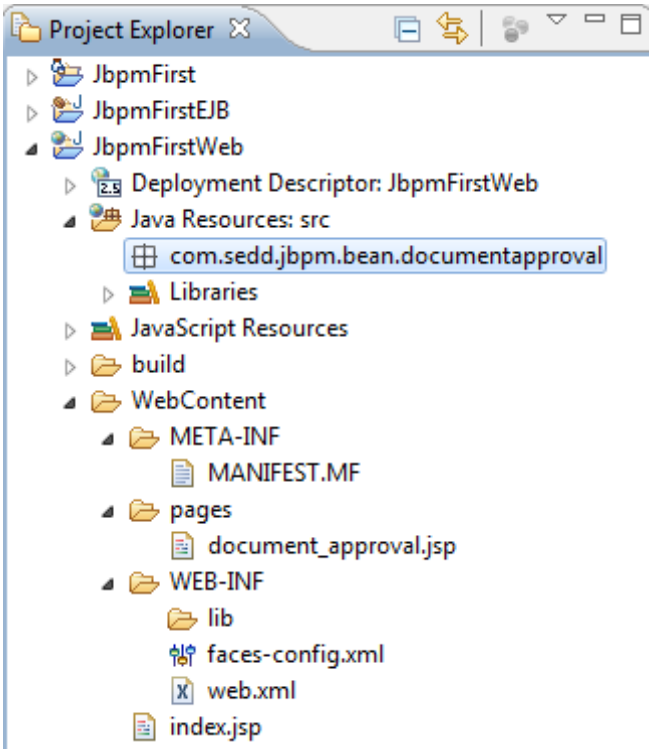




Ignore console errors, edit the file and put the following content in:

```
<?xml version="1.0" encoding="UTF-8"?>

<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
      version="1.2">

</faces-config>
```

The backing beans will be placed under a generic bean package, in business logic dedicated packages.



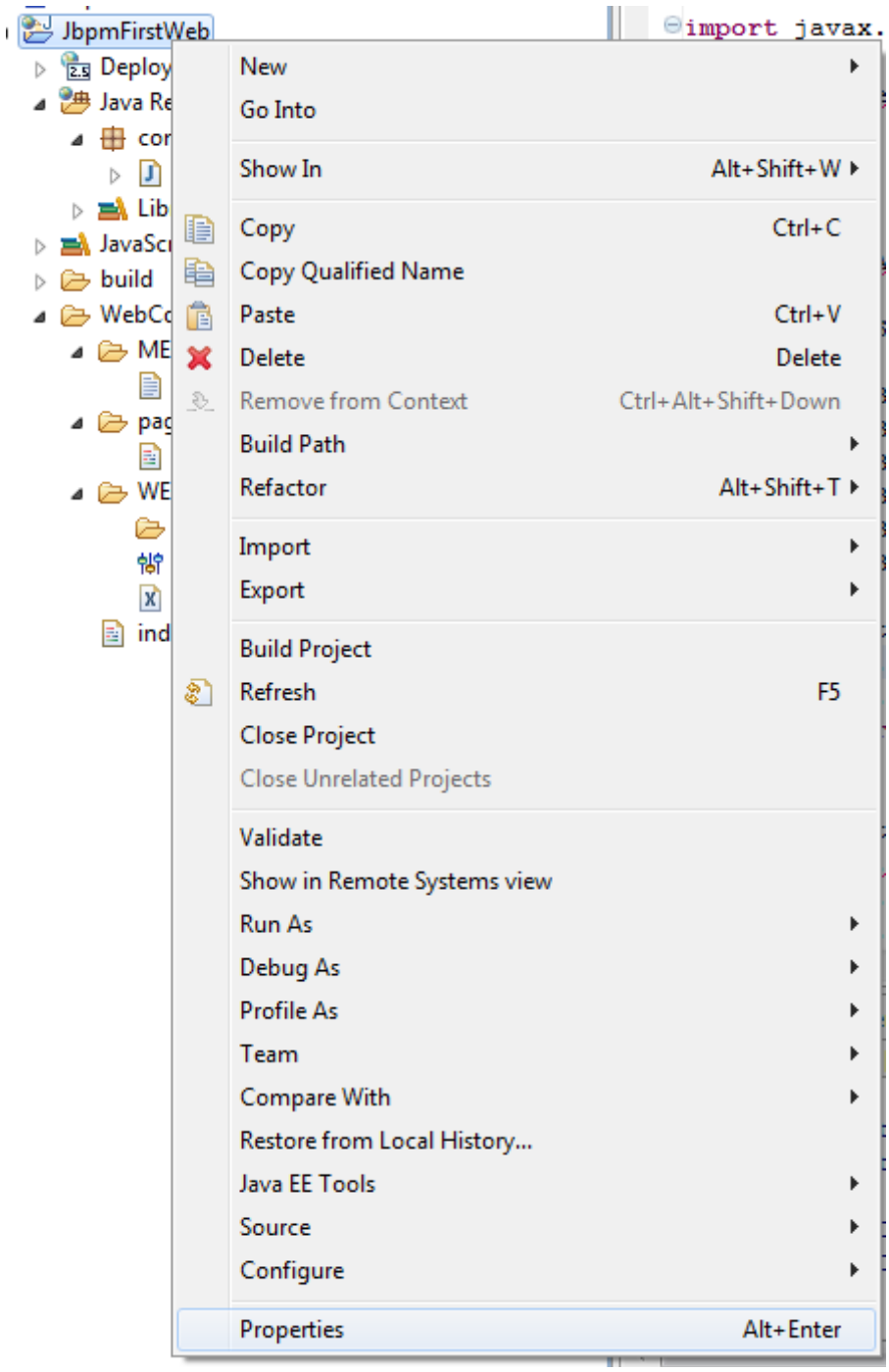
Now create DocumentApproval class with the following content:

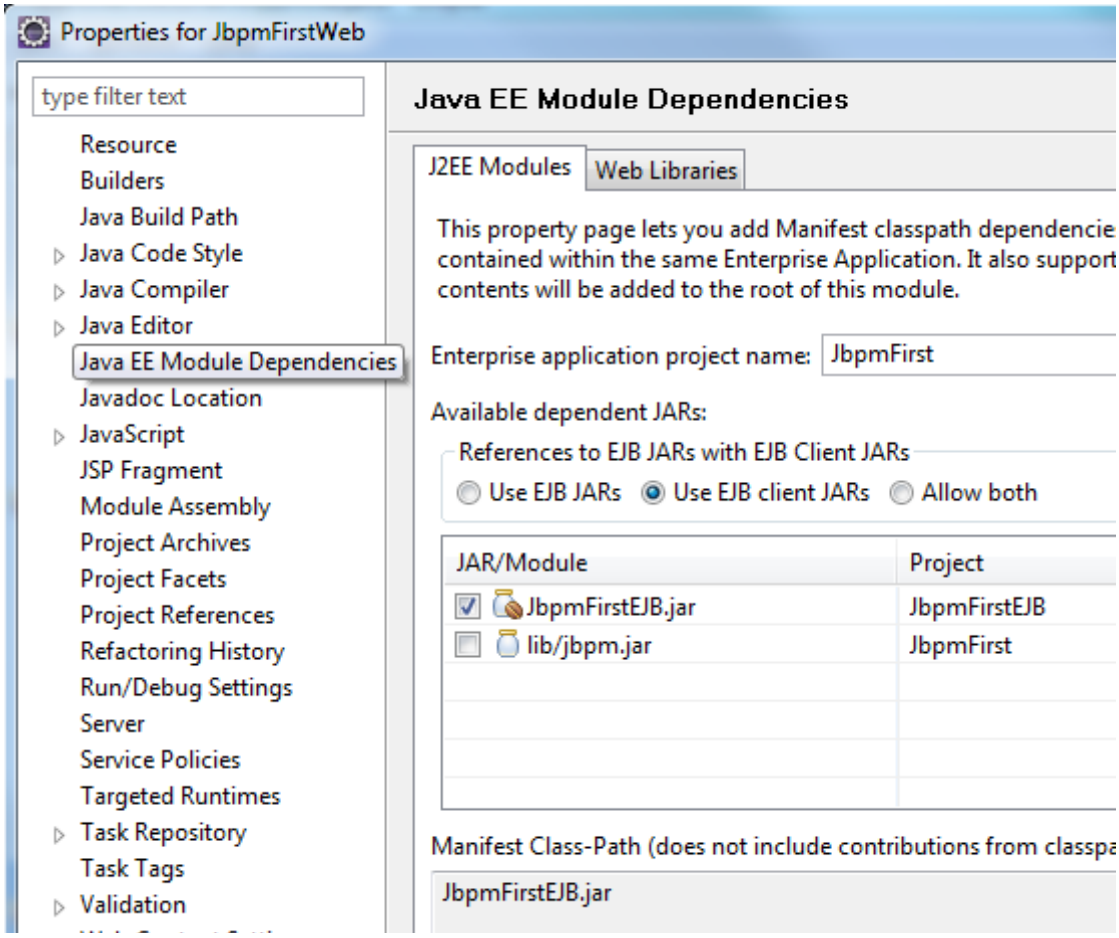
```
package com.sedd.jbpm.bean.documentapproval;

import javax.ejb.EJB;

import com.sedd.jbpm.action.documentapproval.DocumentApprovalBeanLocal;

public class DocumentApproval {

	@EJB
	DocumentApprovalBeanLocal documentApprovalBean;

	private String businessKey;

	private Boolean started = false;
	private Boolean accepted = false;
	private Boolean rejected = false;
	private Boolean checked = false;
	private Boolean submitted = false;
	private Boolean ended = false;

	public String startAction() {
		this.businessKey =
this.documentApprovalBean.startNewDocumentApprovalProcess();
		this.started = this.businessKey != null ? true : false;
		return null;
	}

	public String approveAction() {
		this.documentApprovalBean.documentAccepted(this.businessKey);
		this.accepted = true;
		this.checked = true;
		return null;
	}
```

```
public String rejectAction() {
	this.docummentApprovalBean.documentRejected(this.businessKey);
	this.rejected = true;
	this.checked = true;
	return null;
}

public String submitAction() {
	this.docummentApprovalBean.documentSubmitted(this.businessKey);
	this.submitted = true;
	return null;
}

public String endAction() {

this.docummentApprovalBean.endDocumentApprovalProcess(this.businessKey);
	this.ended = true;
	return null;
}

public String getBusinessKey() {
	return businessKey;
}

public void setBusinessKey(String businessKey) {
	this.businessKey = businessKey;
}

public Boolean getStarted() {
	return started;
}

public void setStarted(Boolean started) {
	this.started = started;
}

public Boolean getAccepted() {
	return accepted;
}

public void setAccepted(Boolean accepted) {
	this.accepted = accepted;
}

public Boolean getRejected() {
	return rejected;
}

public void setRejected(Boolean rejected) {
	this.rejected = rejected;
}

public Boolean getChecked() {
	return checked;
}

public void setChecked(Boolean checked) {
	this.checked = checked;
}

public Boolean getSubmitted() {
	return submitted;
}
```

```
	public void setSubmitted(Boolean submitted) {
		this.submitted = submitted;
	}

	public Boolean getEnded() {
		return ended;
	}

	public void setEnded(Boolean ended) {
		this.ended = ended;
	}

}
```

You will fix the errors by referencing the EJB project from the Web project:

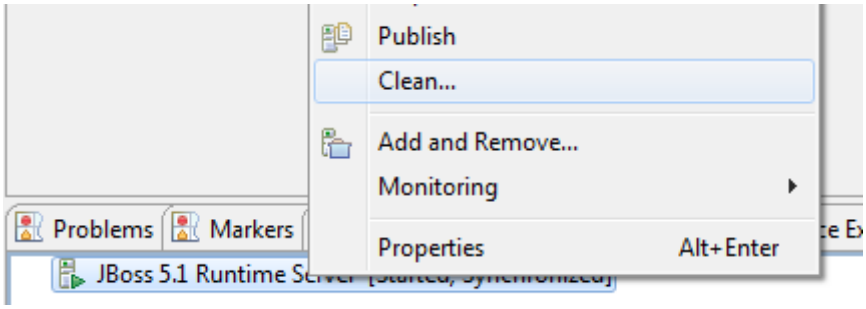

JbpmFirstWeb
import javax.
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import
Export
Build Project
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Java EE Tools
Source
Configure
Properties Alt+Enter



So far so good. The only thing left is to tell JSF about this backing bean. Change the content of faces-config.xml to the following:

```
<?xml version="1.0" encoding="UTF-8"?>

<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
      version="1.2">

      <managed-bean>
            <managed-bean-name>documentApproval</managed-bean-name>
            <managed-bean-
class>com.sedd.jbpm.bean.documentapproval.DocumentApproval</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>

</faces-config>
```

Now do another JBoss clean, and this should be it.



Now browse to http://localhost:8080/JbpmFirstWeb

I'll bet you'll figure out how to test the process and how to follow the console messages.

NOTE: because the backing bean is tied to session, you'll have to do another JBoss clean to start a new test.

HAVE FUN WITH IT!