OpenKM

Workflow guide

PDF generated using the open source mwlib toolkit. See http://code.pediapress.com/ for more information.
PDF generated at: Thu, 25 Oct 2012 10:42:16 UTC

Contents

Articles

References

Workflow Guide

JBoss jBPM is a flexible, extensible framework for process languages. jPDL is one process language that is build on top of that common framework. It is an intuitive process language to express business processes graphically in terms of tasks, wait states for asynchronous communication, timers, automated actions,... To bind these operations together, jPDL has the most powerful and extensible control flow mechanism.

jPDL has minimal dependencies and can be used as easy as using a java library. But it can also be used in environments where extreme throughput is crucial by deploying it on a J2EE clustered application server.

jPDL can be configured with any database and it can be deployed on any application server.

- Overview
- jBPM installation
- jBPM configuration
- Eclipse plugin
  - Eclipse plugin: Installation
  - Eclipse plugin: Usage
- Process modelling 🔒
- Hello world!
- Starting a workflow
- Use of node node 🔒
- Use of decision node 🔒
- Use of task node 🔒
- User input request
- Workflow Forms definition
- Administration interface
- Examples
  - Examples: Simple
  - Examples: Medium
  - Examples: Advanced 🔒
  - Examples: Purchase 🔒
  - Examples: Process Handler 🔒

Let's see a example execution of the medium workflow at Sample Workflow Execution.

Here you have some links with documentation and tutorials:

- jBPM User Guide [1]
- jBPM Tutorial [2]
- jBPM Simulation Tutorial [3]
- JBoss Tools News and Noteworthy [4]

Also there are some interesting books:

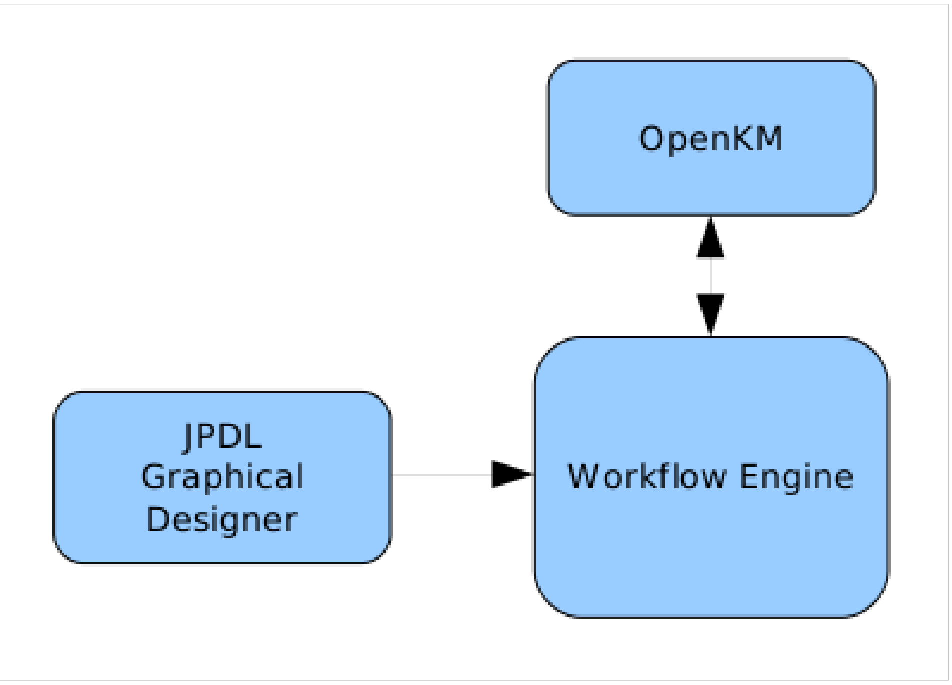
- Business Process Management with JBoss jBPM
- jBPM Developer Guide

References

[1] http://docs.jboss.com/jbpm/v3.2/userguide/html/
[2] http://www.mastertheboss.com/en/jbpm/51-jbpm-tutorial-part.html
[3] http://www.bpm-guide.de/open-source-bpm/jbpm-simulation-tutorial/
[4] http://docs.jboss.org/tools/whatsnew/

Overview

The core workflow and BPM functionality is packaged as a simple java library. This library includes a service to manage and execute processes in the jPDL database.



You can see a working example at jBPM Overview [1].

References

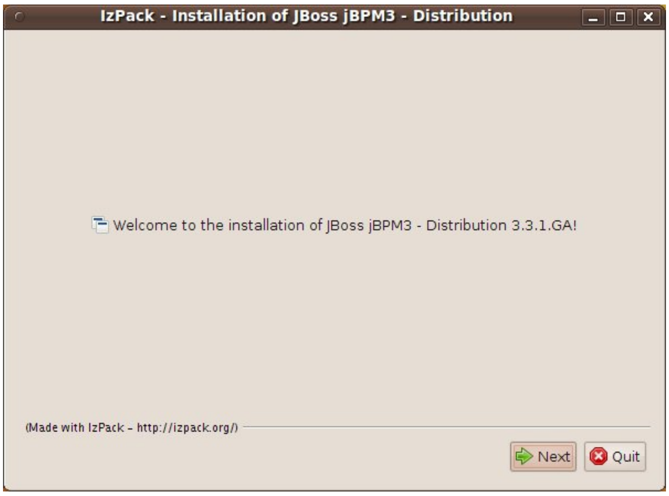
[1] http://docs.jboss.com/jbpm/v3/demos/movies/jbpm-overview.htm

JBPM installation

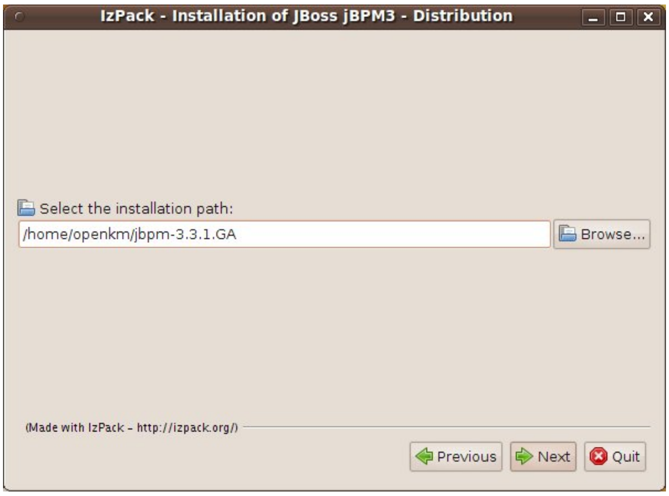
First of all, you need the jBPM libraries installed in your system. There is an installer available at http://sourceforge.net/projects/jbpm/files/. You should download the jBPM-3.3.1.GA version. After that, you can run the install process:

```
$ java -jar jbpm-installer-3.3.1.GA.jar
```

You have to select a destination path where the library will be installed:

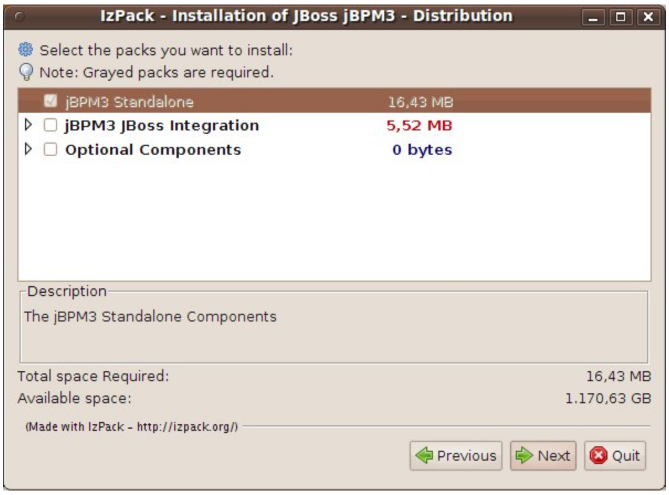

IzPack - Installation of JBoss jBPM3 - Distribution
(Made with IzPack - http://izpack.org/)
Next
Quit

Welcome to the installation of JBoss jBPM3 - Distribution 3.3.1.GA!


IzPack - Installation of JBoss jBPM3 - Distribution
Browse...
(Made with IzPack - http://izpack.org/)
Previous
Next
Quit

Select the installation path: /home/openkm/jbpm-3.3.1.GA

And can only select the minimun requirements to install, because we only need the jBPM library:

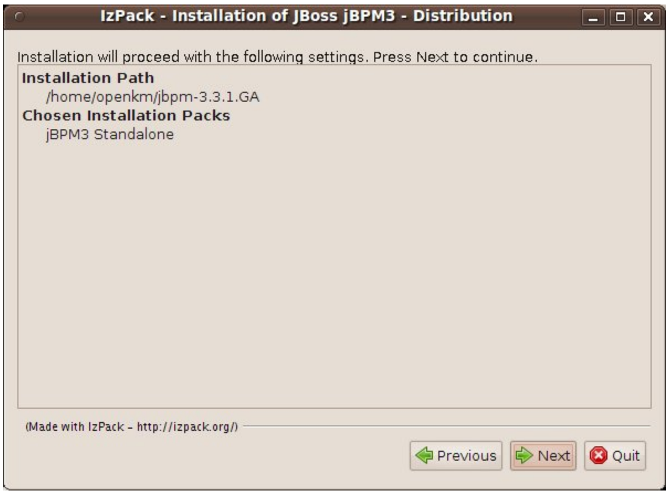

IzPack - Installation of JBoss jBPM3 - Distribution
(Made with IzPack - http://izpack.org/)
Previous
Next
Quit

Select the packs you want to install:
Note: Grayed packs are required.

☑	jBPM3 Standalone	16,43 MB
☐	**jBPM3 JBoss Integration**	**5,52 MB**
☐	**Optional Components**	**0 bytes**

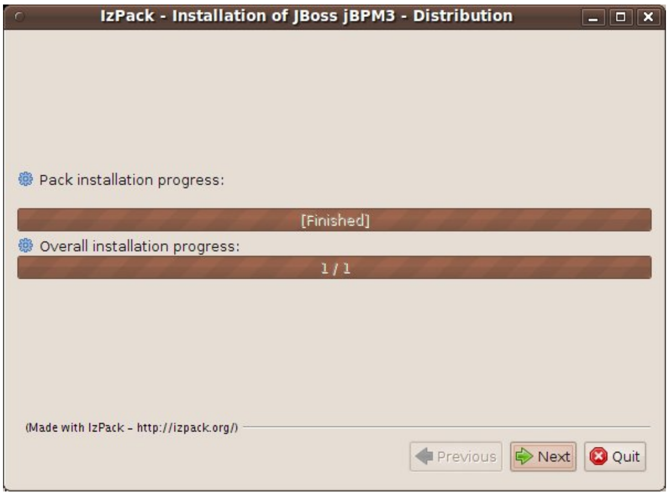
Description
The jBPM3 Standalone Components

Total space Required: 16,43 MB
Available space: 1.170,63 GB

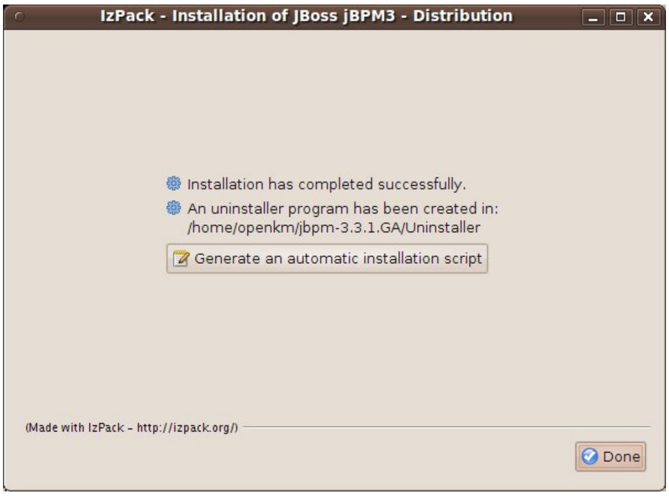

IzPack - Installation of JBoss jBPM3 - Distribution
(Made with IzPack - http://izpack.org/)
Previous
Next
Quit

Installation will proceed with the following settings. Press Next to continue.
Installation Path
/home/openkm/jbpm-3.3.1.GA
Chosen Installation Packs
jBPM3 Standalone

When finished, the libraries needed to implement workflows are copied at */home/openkm/jbpm-3.3.1.GA*. Remember this path because you need to tell to Eclipse where these libraries are located.



IzPack - Installation of JBoss jBPM3 - Distribution
Pack installation progress:
[Finished]
Overall installation progress:
1 / 1
(Made with IzPack - http://izpack.org/)
Previous
Next
Quit



Installation has completed successfully.

An uninstaller program has been created in: /home/openkm/jbpm-3.3.1.GA/Uninstaller

IzPack - Installation of JBoss jBPM3 - Distribution
Generate an automatic installation script
(Made with IzPack - http://izpack.org/)
Done

JBPM configuration

jBPM configuration file is located at $TOMCAT_HOME/jbpm.cfg.xml. If you edit this file, you will see something like this:

```
<jbpm-configuration>
  <string name="jbpm.mail.smtp.host" value="smtp.your-domain.com " />
  <string name="jbpm.mail.from.address" value="noreply@your-domain.com" />
  <string name="resource.mail.templates" value="jbpm.mail.templates.xml"/>
  <bean name="jbpm.mail.address.resolver" class="com.openkm.workflow.AddressResolver" singleton="true"/>
</jbpm-configuration>
```

Starting with OpenKM 5.1.8 you can also configure jBPM creating a file $JBOSS_HOME/jbpm.xml.

In older OpenKM installation with jboss the path is $JBOSS_HOME/server/default/deploy/OpenKM.war/WEB-INF/classes/jbpm.cfg.xml

As in JBoss mail configuration, you have to edit a couple of properties:

- **jbpm.mail.smtp.host**: this is the host where is located your mail server. Can be localhost if you have a local mail server installed (like Postfix) Typically the same value of mail.smtp.host.
- **jbpm.mail.from.address**: all the mails send by OpenKM will be from this email address. Can be in the form of noreply@your-domain.com. Typically the same value of mail.from.

jBPM will notify you when a task has been assigned to you or when it wants to send you a reminder. These mail templates are defined in the file jbpm.mail.templates.xml which is located in the same place as the previous one. Here you can define a more elegant mail message. Also you have to configure a parameter here:

- **BaseTaskListURL**: a value in the form of http://your-domain.com:8080/OpenKM.

For more info, read jBPM User Guide: Email Support [1].

References

[1] http://docs.jboss.com/jbpm/v3.2/userguide/html/ch15.html

Eclipse plugin

jPDL also includes a graphical designer tool. The designer is a graphical tool for authoring business processes. It's an eclipse plugin.

The most important feature of the graphical designer tool is that it includes support for both the business analyst as well as the technical developer. This enables a smooth transition from business process modeling to the practical implementation.

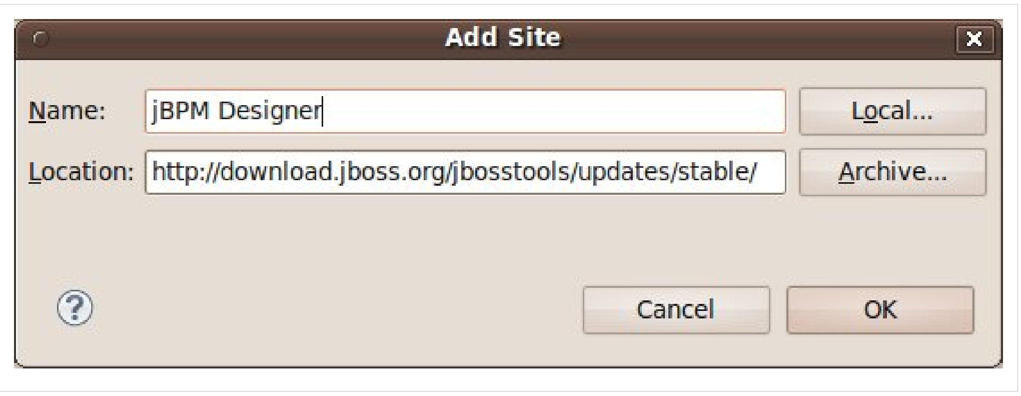
- Eclipse plugin: Installation
- Eclipse plugin: Usage

Eclipse plugin: Installation

First of all, download Eclipse Galileo from http://www.eclipse.org/galileo/. Get the **Eclipse IDE for Java Developers** version. Once you have installed the application, start it and will install the jBPM designed.

A more recent Eclipse called Helios has been released. You can download it from [1]

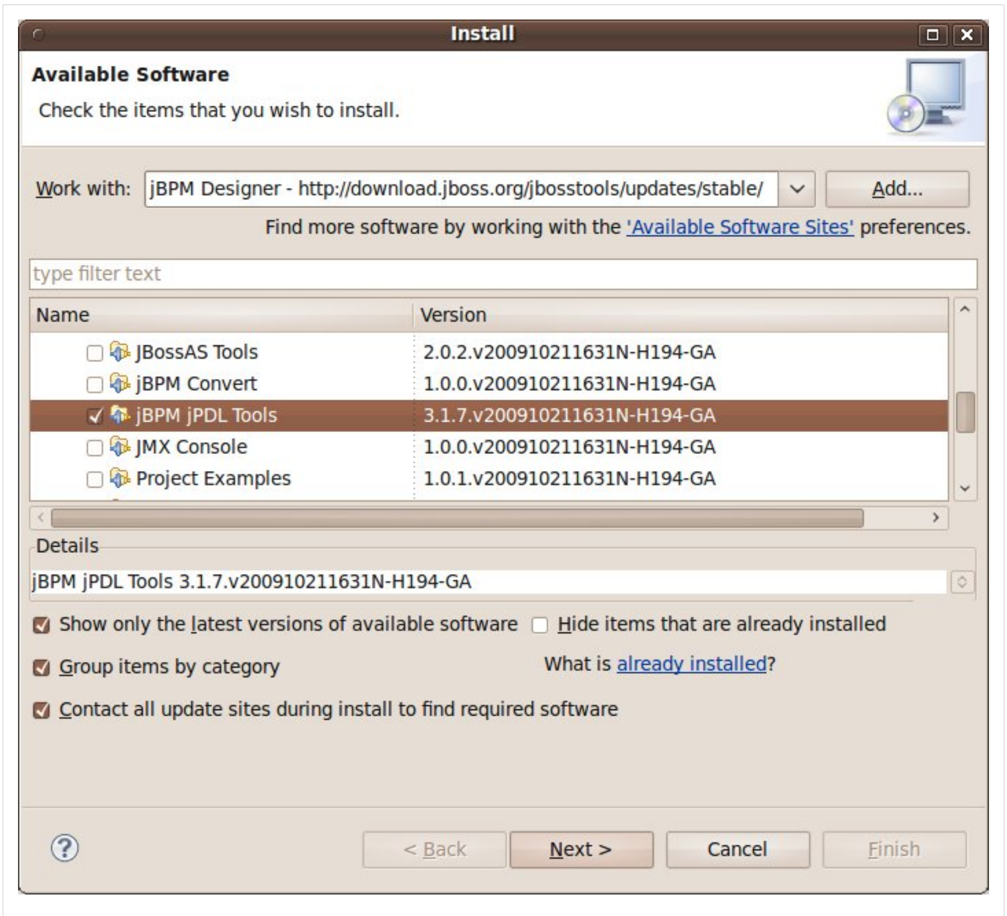
Go to Help → Install New Software... Add a new site with this location http://download.jboss.org/jbosstools/updates/stable/:



Name: jBPM Designer

Location: http://download.jboss.org/jbosstools/updates/stable/

Now select **jBPM jPDL Tools** from the list and click on Next button.



Available Software

Check the items that you wish to install.

Work with: jBPM Designer - http://download.jboss.org/jbosstools/updates/stable/

Find more software by working with the 'Available Software Sites' preferences.

type filter text

Name	Version
☐ JBossAS Tools	2.0.2.v200910211631N-H194-GA
☐ jBPM Convert	1.0.0.v200910211631N-H194-GA
☑ jBPM jPDL Tools	3.1.7.v200910211631N-H194-GA
☐ JMX Console	1.0.0.v200910211631N-H194-GA
☐ Project Examples	1.0.1.v200910211631N-H194-GA

Details

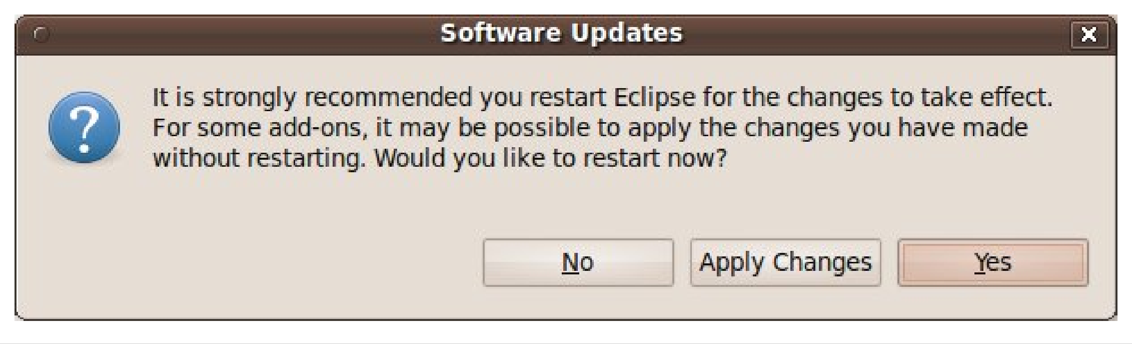
jBPM jPDL Tools 3.1.7.v200910211631N-H194-GA

☑ Show only the latest versions of available software ☐ Hide items that are already installed

☑ Group items by category What is already installed?

☑ Contact all update sites during install to find required software

Eclipse should be restarted to finalize the plugin installation.



It is strongly recommended you restart Eclipse for the changes to take effect. For some add-ons, it may be possible to apply the changes you have made without restarting. Would you like to restart now?

Eclipse Helios & JBoss Tools 3.2

Recently the packages have been refactorized and is located at **All JBoss Tools 3.2.0 > jBPM 3 Tools Runtime**.

Eclipse Indigo & JBoss Tools 3.3

If you use Eclipse Indigo, follow instructions from JBoss Tools 3.3 Installation From Update Site [2]. You have two available software sites:

- http://download.jboss.org/jbosstools/updates/development/indigo/
- http://download.jboss.org/jbosstools/updates/development/indigo/soa-tooling/

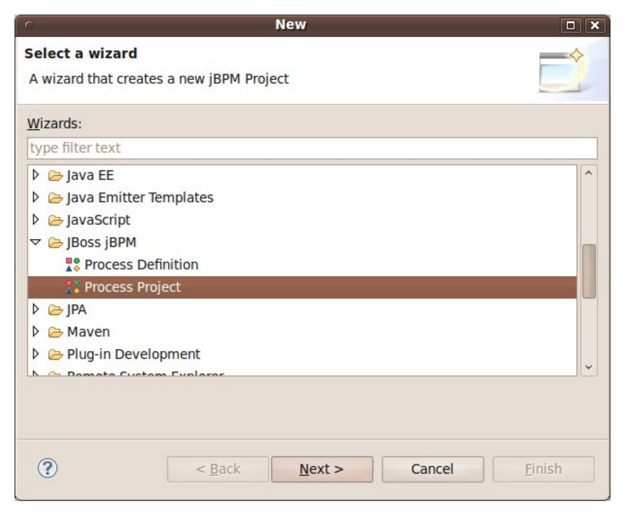
The jBPM Graphical Editor is located in the **SOA Tooling** source. Once registered the site, check the option at **All JBoss Tools - SOA Tooling 3.3 > jBPM 3 Tools Runtime**.

References

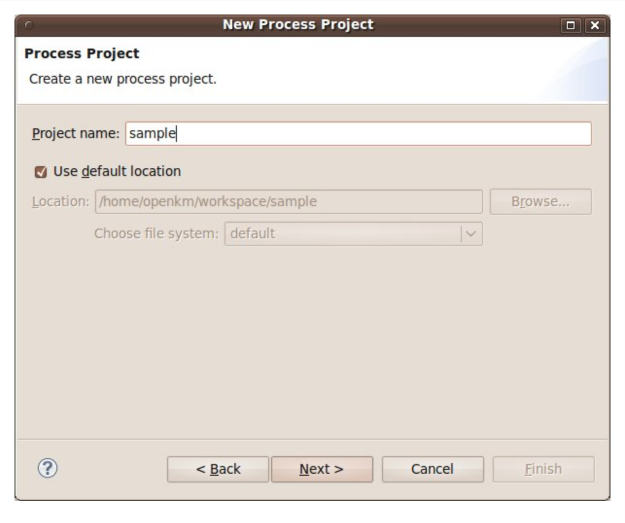
[1] http://www.eclipse.org/downloads/.
[2] http://www.jboss.org/tools/download/installation/update_3_3

Eclipse plugin: Usage

To create a new process definition, go to File → New → Other and select "Process Definition", but before you need to create a "Process Project". Will appear another form where you need to specify the source folder and the process name. Lets call it simply "hello".



A wizard that creates a new jBPM Project



Create a new process project.

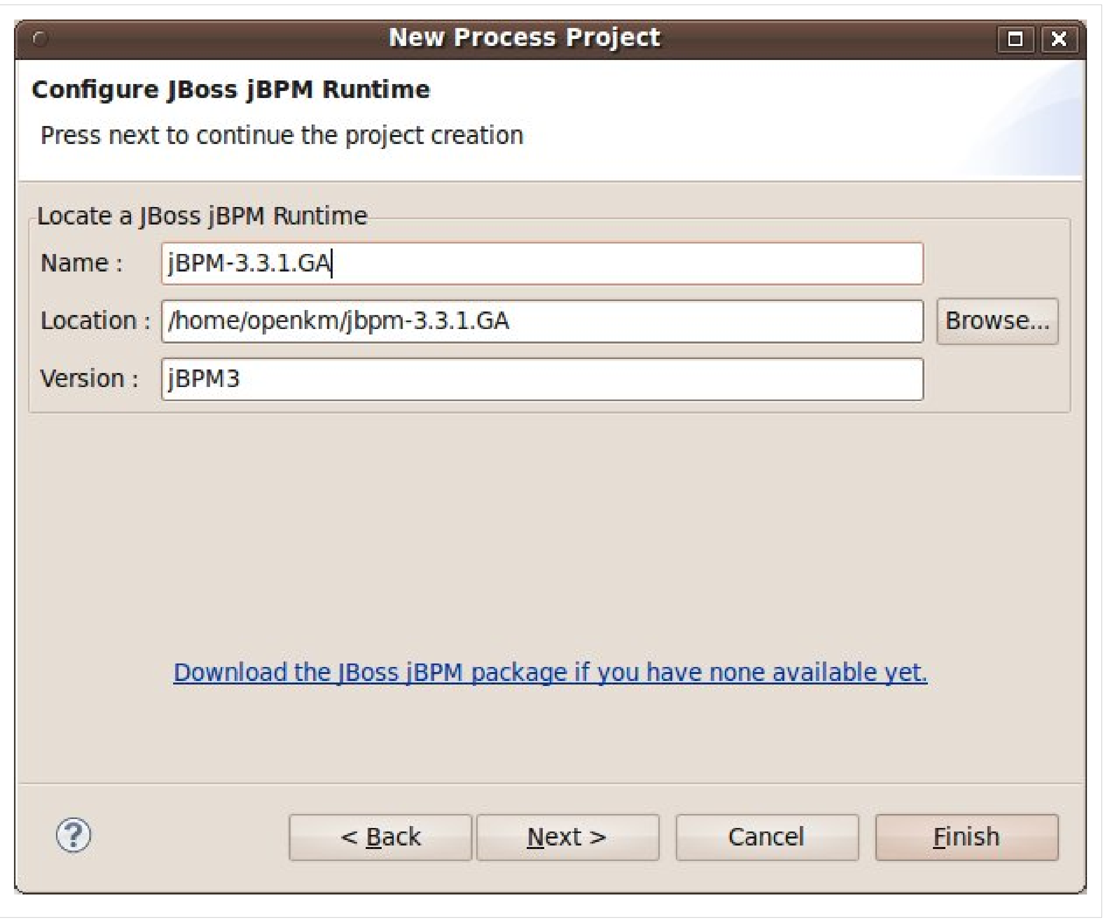
Project name: sample

☑ Use default location

Location: /home/openkm/workspace/sample

Choose file system: default

If is the first time you create a Process Definition, you have to configure the jBPM runtime. Remember the path of the jBPM library installation (See JBPM installation)



Configure JBoss jBPM Runtime

Press next to continue the project creation

Locate a JBoss jBPM Runtime

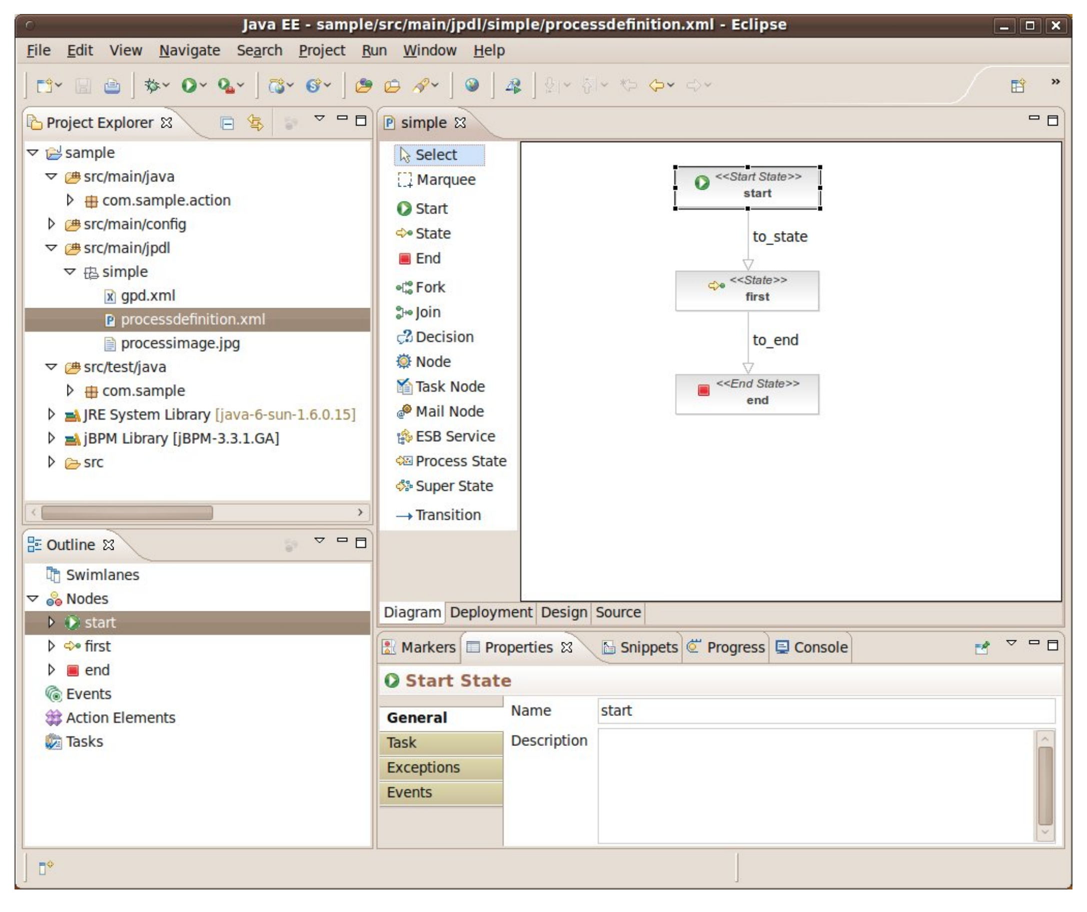
Name : jBPM-3.3.1.GA

Location : /home/openkm/jbpm-3.3.1.GA

Version : jBPM3

Browse...
Download the JBoss jBPM package if you have none available yet.
< Back
Next >
Cancel
Finish

Once the wizard has finished, you can see an Eclipse environment with a central panel divided in four tabs:

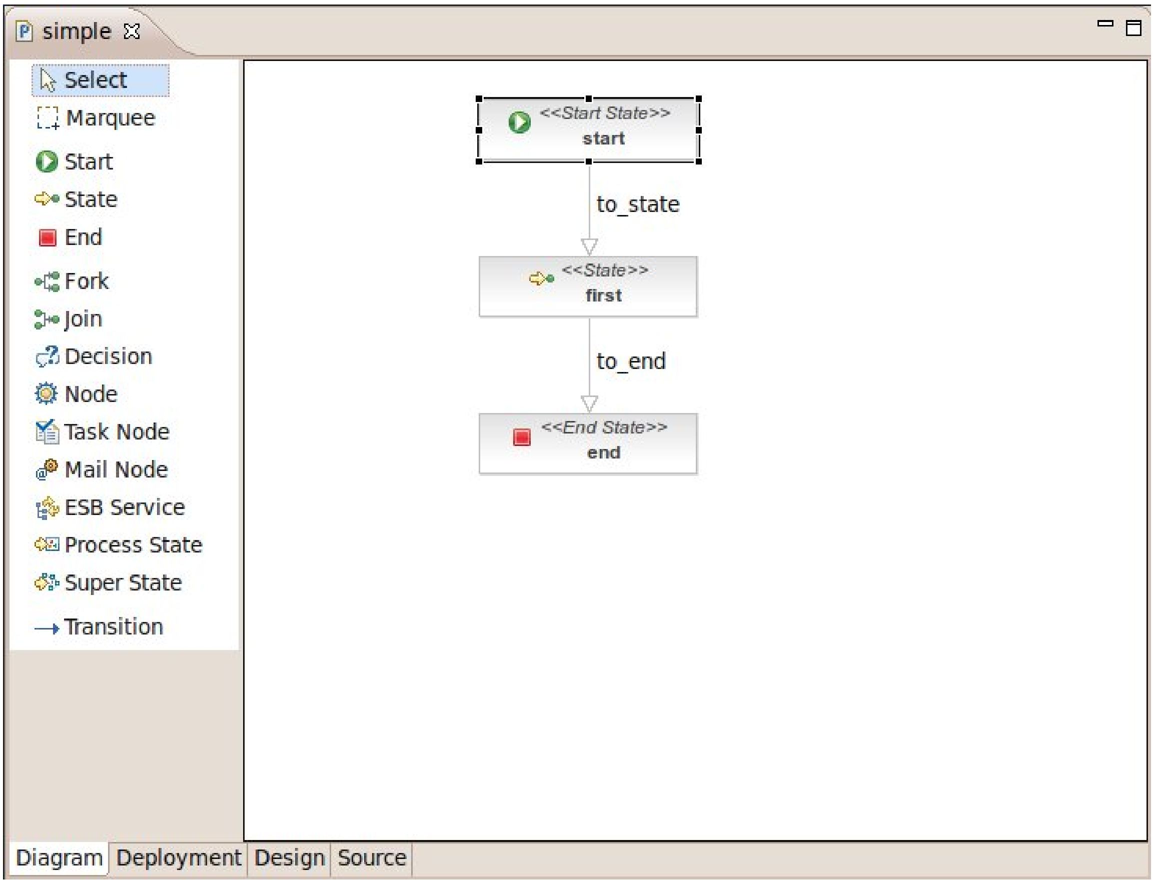


Name: start

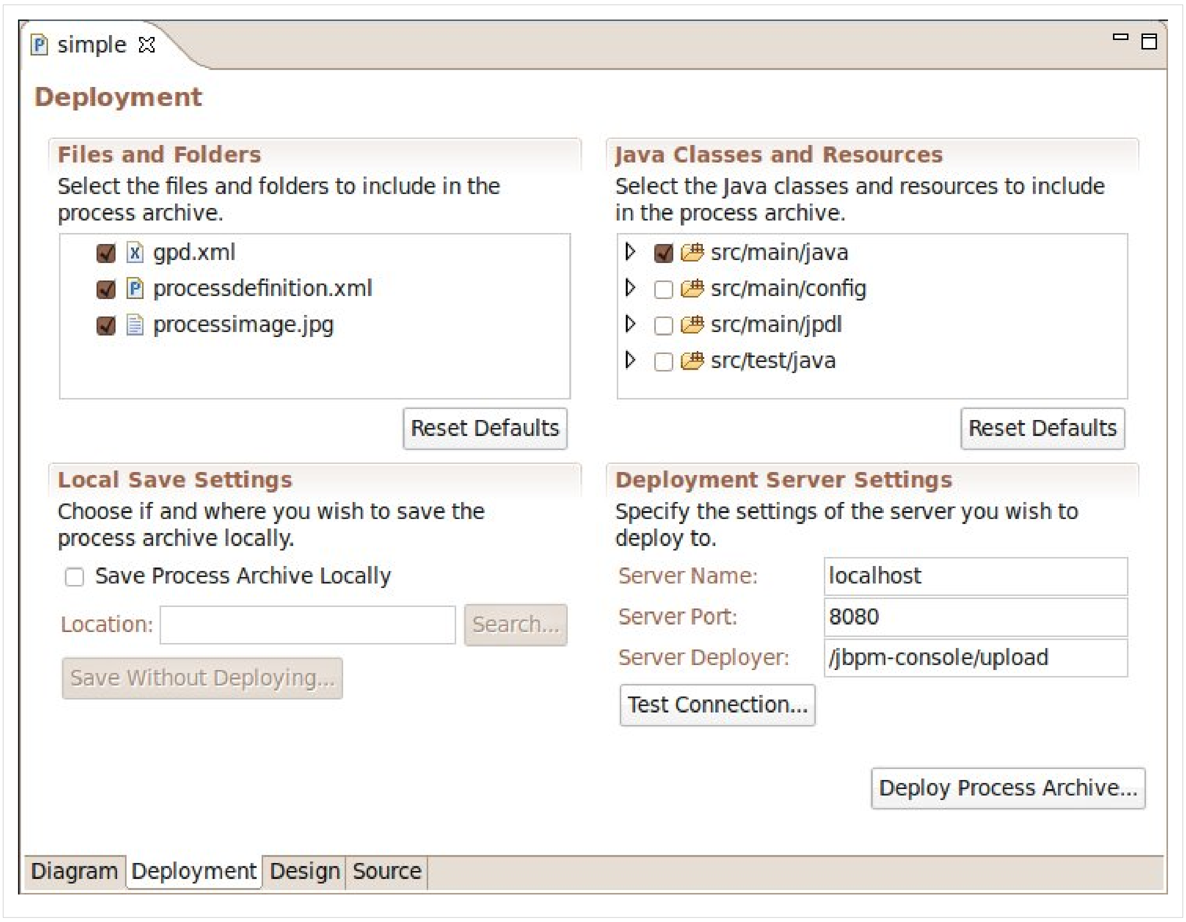
There is a new revision of the jBPM Eclipse plugin with some changes. They are described at:

- jBPM Tools 3.2.0.M2 What's New [1]
- jBPM Tools 3.2.0.M1 What's New [2]
- jBPM Tools 3.3.0.M2 What's New [3]

- **Diagram**: This is the main work zone. Here you can construct your process definitions on a visual way.



- **Deployment**: Here you can deploy the created process definition directly to the server or save it to deploy manually. You can create a deployable archive going to "Local Save Settings" section, marking the "Save Process Archive Locally" option, selecting a location and clicking in the "Save Without Deploying..." button. The file extension should be ".par".



Files and Folders

Select the files and folders to include in the process archive.

- ☑ gpd.xml
- ☑ processdefinition.xml
- ☑ processimage.jpg

Java Classes and Resources

Select the Java classes and resources to include in the process archive.

- ☑ src/main/java
- ☐ src/main/config
- ☐ src/main/jpdl
- ☐ src/test/java

Local Save Settings

Choose if and where you wish to save the process archive locally.

☐ Save Process Archive Locally

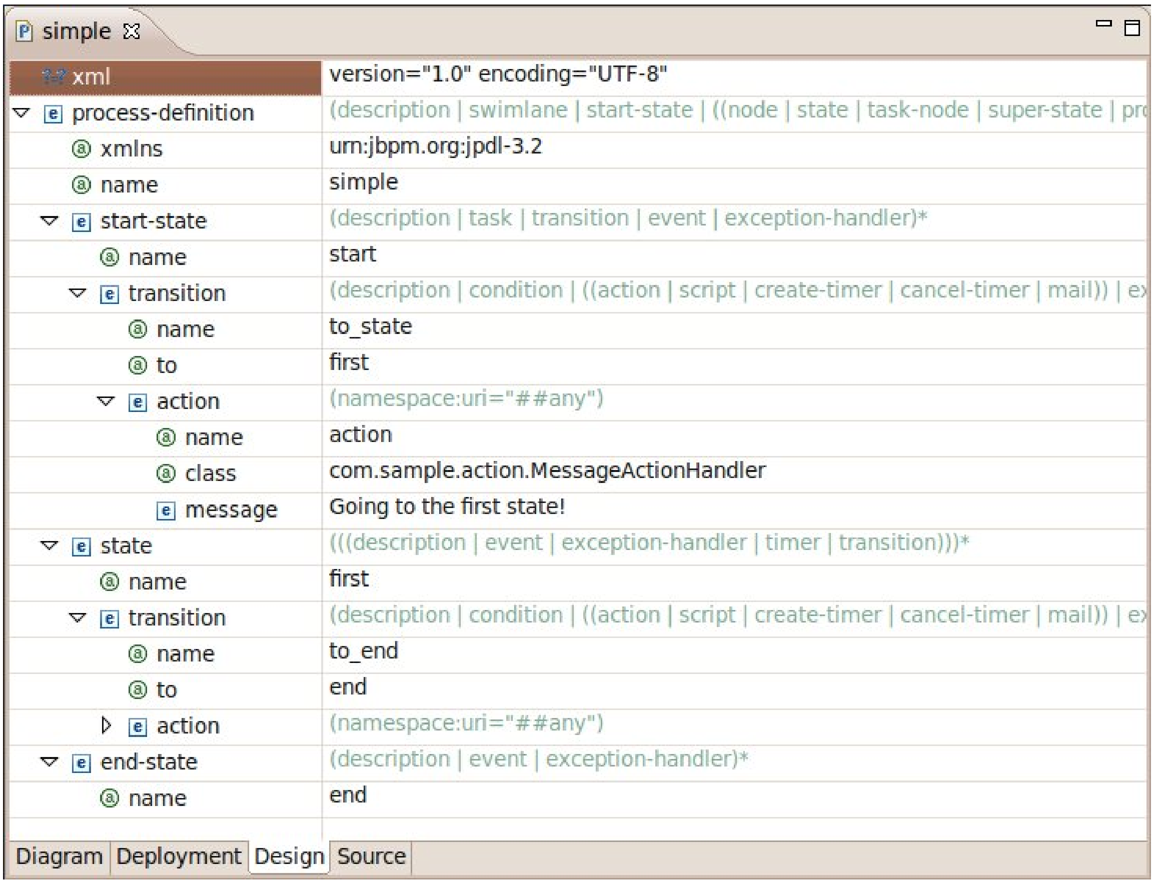
Location:

Deployment Server Settings

Specify the settings of the server you wish to deploy to.

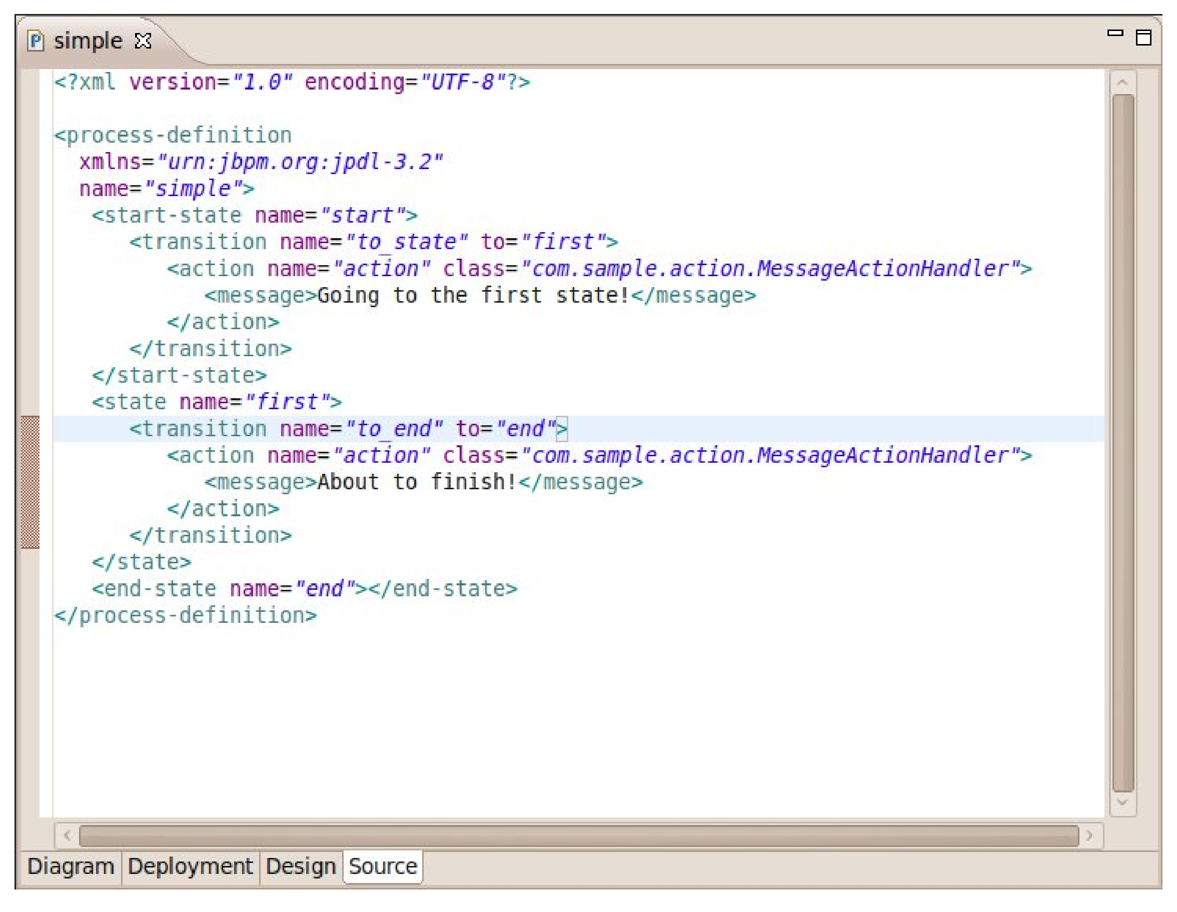
Server Name: localhost
Server Port: 8080
Server Deployer: /jbpm-console/upload

- **Design**: Is the source code of the process definition. It is an hierarchical representation of the XML which describe the nodes and transitions. See jPDL xml schema chapter in jBPM jPDL User Guide for more info.



xml	version="1.0" encoding="UTF-8"
process-definition	(description | swimlane | start-state | ((node | state | task-node | super-state | pr
xmlns	urn:jbpm.org:jpdl-3.2
name	simple
start-state	(description | task | transition | event | exception-handler)*
name	start
transition	(description | condition | ((action | script | create-timer | cancel-timer | mail)) | e
name	to_state
to	first
action	(namespace:uri="##any")
name	action
class	com.sample.action.MessageActionHandler
message	Going to the first state!
state	(((description | event | exception-handler | timer | transition))*
name	first
transition	(description | condition | ((action | script | create-timer | cancel-timer | mail)) | e
name	to_end
to	end
action	(namespace:uri="##any")
end-state	(description | event | exception-handler)*
name	end

- **Source**: Is the source code of the process definition. It is an XML which describe the nodes and transitions. See jPDL xml schema chapter in jBPM jPDL User Guide for more info.



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

<process-definition
  xmlns="urn:jbpm.org:jpdl-3.2"
  name="simple">
   <start-state name="start">
      <transition name="to_state" to="first">
         <action name="action" class="com.sample.action.MessageActionHandler">
            <message>Going to the first state!</message>
         </action>
      </transition>
   </start-state>
   <state name="first">
      <transition name="to_end" to="end">
         <action name="action" class="com.sample.action.MessageActionHandler">
            <message>About to finish!</message>
         </action>
      </transition>
   </state>
   <end-state name="end"></end-state>
</process-definition>
```

See also:

- jBPM Tools Reference Guide [4]

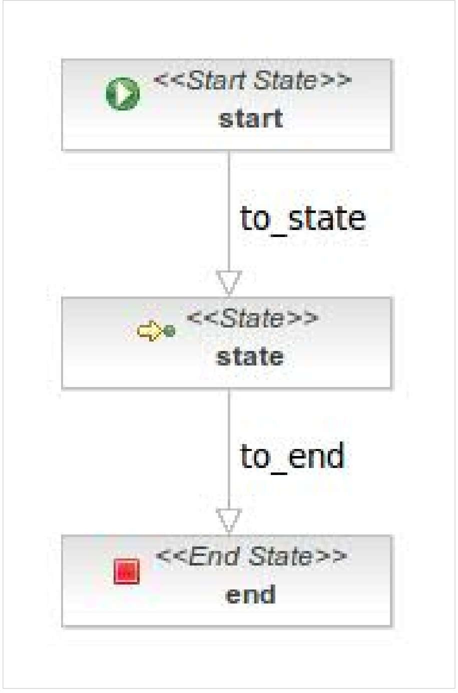
References

[1] http://docs.jboss.org/tools/whatsnew/jbpm/jbpm-news-3.2.0.M2.html
[2] http://docs.jboss.org/tools/whatsnew/jbpm/jbpm-news-3.2.0.M1.html
[3] http://docs.jboss.org/tools/whatsnew/jbpm/jbpm-news-3.3.0.M2.html
[4] http://www.redhat.com/developer_studio/guides/jbpm/html_single/

Process modelling

We are going to describe the elements which define a jBPM process. A process is composed by:

- Nodes
- Transitions
- Actions



To create a process definition in a graphical way, you can use jBPM Graphical Process Designer. This tool is packaged as a Eclipse plugin.

For a more detailed description, please read jBPM User Guide: Process Modeling [1]. Also is recommended to read:

- JBPM best practices [2]
- JBPM Mail delivery [3]

Nodes

The define the states of the **process definition**. They are connected by transitions. Both define the different path which can be followed in a running process definition. A running process definition is called **process instance**.

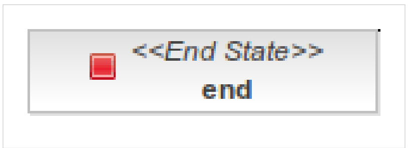
There are several types of nodes:

Start node

The start node defined the process entry point. Only one start node is permitted.

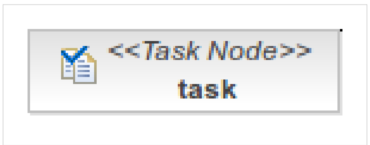
End node

End nodes define the end of the process execution. A process may have several end nodes. In this case the process finish when the process arrives to any of these end nodes.



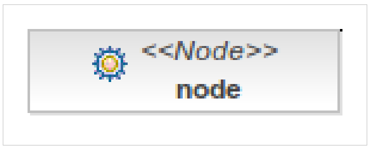
Task node

A task node represents one or more tasks that are to be performed by humans.



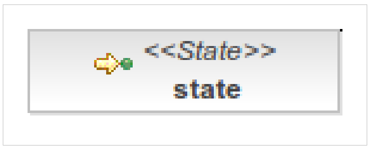
Node node

You can define the behaviour using a Action element, which will be executed when the process arrives to the node.



State node

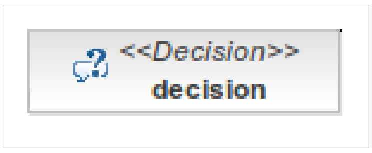
A state is a bare-bones wait state. The difference with a task node is that no task instances will be created in any task list. This can be usefull if the process should wait for an external system.



Decision node

There are 2 ways to model a decision. The distinction between the two is based on *who* is making the decision.

- When the decision is to be taken by the process, a decision node should be used.
- When the decision is taken by an external party, you should use multiple transitions leaving a state or wait state node.



Fork node

A fork node splits one path of execution into multiple concurrent paths of execution.

Join node

The join node take all these concurrent executions before continue with the process execution.

Transitions

Transitions have a source node and a destination node. The source node is represented with the property from and the destination node is represented by the property to. A transition can optionally have a name.

Actions

Actions are pieces of java code that are executed upon events in the process execution. The main event types are entering a node, leaving a node and taking a transition.

Note the difference between an action that is placed in an event versus an action that is placed in a node:

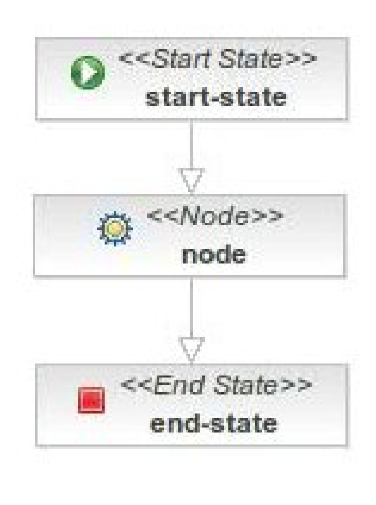
- Actions that are put in an event are executed when the event fires. Actions on events have no way to influence the flow of control of the process.
- An action that is put on a node has the responsibility of propagating the execution.

References

[1] http://docs.jboss.com/jbpm/v3.2/userguide/html/ch09.html
[2] http://www.mastertheboss.com/jbpm/106-jbpm-best-practices.html
[3] http://www.mastertheboss.com/jbpm/75-jbpm-mail.html

Hello world!

A process definition is a directed graph, made up of nodes and transitions. The hello world process has 3 nodes. To see how the pieces fit together, we're going to start with a simple process without the use of the designer tool. The following picture shows the graphical representation of the hello world process:



And this is the XML which generates this process definition graph:

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="" name="Hello world">
  <start-state name="start-state">
    <transition to="node"></transition>
  </start-state>
  <node name="node">
    <transition to="end-state"></transition>
  </node>
  <end-state name="end-state"></end-state>
</process-definition>
```

This is a very basic example of a process definition. Is useless unless you make some improvements like adding an action.

Once the process definition is created, you can register it in the workflow engine. Log as okmAdmin and go to Administration → Workflow. The process definition list is actually empty. To register this new process definition go to Eclipse and to the Deployment tab. The Server Deployer input should be set to /OpenKM/upload. Click on "Deploy Process Archive..." button.

If you go to the process definition list in the OpenKM administration, you can click on the "Reload" link and the new process will appear on the list. Here you can also upload a process definition archive, if you want.

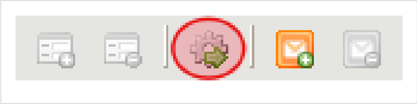
If you register a process definition several times, the old definition persist until removed. This mean that will be several versions of a process definition. This is because you can update a process definition but the old one may be executing actually.

When you start a workflow related to a document, it always will use the last version of the process definition.

Starting a workflow

You have created a process definition and registered in the engine. Now will see how to start a workflow.

From the OpenKM web interface, access to the Desktop (the default tab when you log into OpenKM). Here you can see the documents and folders stored in the repository. To start an workflow from OpenKM you must select a document and push on the button in the toolbar.



In OpenKM all the workflows are related to a document or folder.

In this simple case, the workflow will begin and end almost instantly because there is no request for user to input data. In the next sample will request some data to the user.

If you have registered an workflow and it does not appear in the list, you need to add this workflow in the User Profile [1].

References

[1] http://wiki.openkm.com/index.php/Profile

Use of node node

You can define the behavior of a node of type node suing the Action element. This Action is executed when the process arrives to the node.

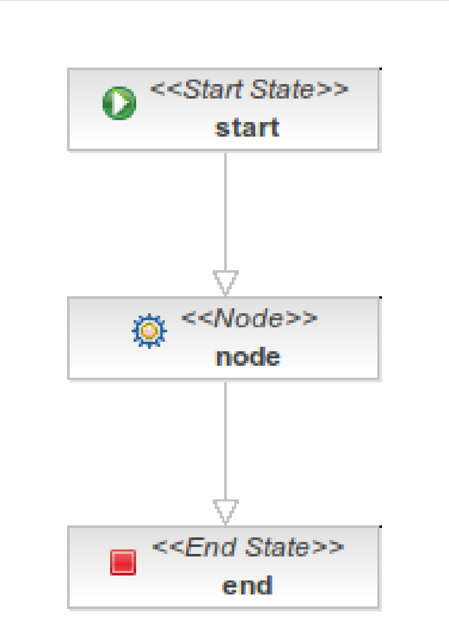
The Action to be executed can be defined in two ways:

- Using a BeanShell script.
- Using an action handler, this is a Java class which implements the ActionHandler [1] interface.

Below we are going to create a sample process with a *node* node which make use of an implementation of the ActionHandler [1] interface.

Source Eclipse project at .

In this image you can see the process definition graph:



This process begins in a start node and go to a *node* node where the action to be performed is defined by the class *MyAction* which implements the ActionHandler [1] interface.

```
public class MyAction implements ActionHandler {
    private static final long serialVersionUID = 1L;

    @Override
    public void execute(ExecutionContext executionContext) throws
Exception {
        System.out.println("Executing programmed action...");

        // Go to next node
        executionContext.getToken().signal();
    }
}
```

The last line of this action tell the jBPM engine to go to the next node, which in this case is the end node.

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="node node">
    <start-state name="start">
        <transition to="node"></transition>
    </start-state>
    <node name="node">
        <action class="com.openkm.sample.MyAction"></action>
        <transition to="end"></transition>
    </node>
   <end-state name="end"></end-state>
</process-definition>
```

References

[1] http://docs.jboss.com/jbpm/v3.2/javadoc-jpdl/org/jbpm/graph/def/ActionHandler.html

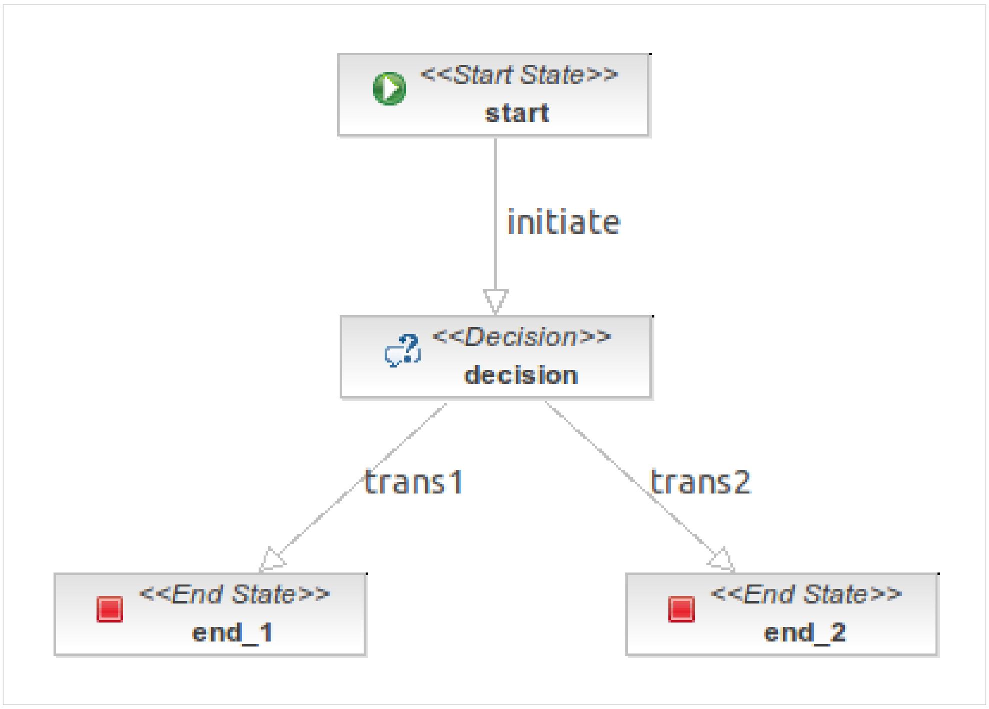
Use of decision node

A decision node is used to guide the process running path. This means you can tell the workflow which way should go. Decision criteria can be specified as follows:

- Adding conditions to transitions or BeanShell script which returns a boolean. Nodes go across its transitions examining the conditions until found the first one which match a criteria.
- Creating a class which implements the DecisionHandler [1] interface. The *decide* method returns the transition to follow.

Source Eclipse project at .

Below we are going to create a process containing a decision node which implements the DecisionHandler [1] interface to decide the next node in the workflow.



This process begins at the start node and go across the transition name "initiate". This transition has an action called "createvalue" handler by the class **ValueAction**, who set the variable "value" in the execution context.

```
public class ValueAction implements ActionHandler {
    private static final long serialVersionUID = 1L;
    public String value;

    @Override
    public void execute(ExecutionContext executionContext) throws
Exception {
        executionContext.getContextInstance().setVariable("value",
value);
    }
}
```

After that arrives to the decision node handled by the class **Decission**, who check the existence of the "value" variable in the execution context. If exists this variable leave the node across the transition "trans1", otherwise go across "trans2".

```
public class MyDecision implements DecisionHandler {
    private static final long serialVersionUID = 1L;

    public MyDecision(String info) {
        super();
    }

    @Override
    public String decide(ExecutionContext executionContext) throws
Exception {
        String value =
(String)executionContext.getContextInstance().getVariable("value");

        if(value != null) {
            return "trans1";
        } else {
            return "trans2";
        }
    }
}
```

Finally when arrives to any of the two end nodes, an action is executed which shows the node where the execution flow ends. This action is handled by the class **ShowMessageAction**:

```
public class ShowMessageAction implements ActionHandler {
    private static final long serialVersionUID = 1L;

    public ShowMessageAction(String info) {
        super();
    }

    @Override
    public void execute(ExecutionContext executionContext) throws
Exception {
        System.out.println("Flow go to node: " +
executionContext.getNode().getName());
    }
}
```

If in the configuration you have set the "value" variable in the console you will see a message saying that has finished in node "end_1". In the other side, will end in node "end_2".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="decision node">
    <start-state name="start">
        <transition to="decision" name="initiate">
            <action name="createvalue" class="com.openkm.sample.ValueAction"></action>
        </transition>
    </start-state>

    <decision name="decision">
        <handler class="com.openkm.sample.MyDecision"></handler>
        <transition to="end_1" name="trans1"></transition>
        <transition to="end_2" name="trans2"></transition>
    </decision>

    <end-state name="end_1">
        <event type="node-enter">
            <action class="com.openkm.sample.ShowMessageAction"></action>
        </event>
    </end-state>

    <end-state name="end_2">
        <event type="node-enter">
            <action class="com.openkm.sample.ShowMessageAction"></action>
        </event>
    </end-state>
</process-definition>
```

References

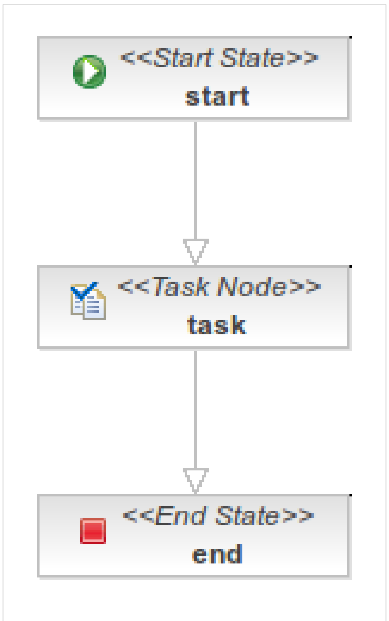
[1] http://docs.jboss.com/jbpm/v3.2/javadoc-jpdl/org/jbpm/graph/node/DecisionHandler.html

Use of task node

Now we are going to see how you can create task in jBPM and assign them to actors who will perfom the task. Actors also can select a pending task and assign to himself.

Source Eclipse project at .

First of let's create a process definition called "task" with an initial node, a task node and a end node:



As seen previously, a task node represents one or many tasks executed by a human person. When the process execution arrives to a task node, an task instance is created in the workflow member list. This node will enter in a wait state until the user notify the conclusion of the task.

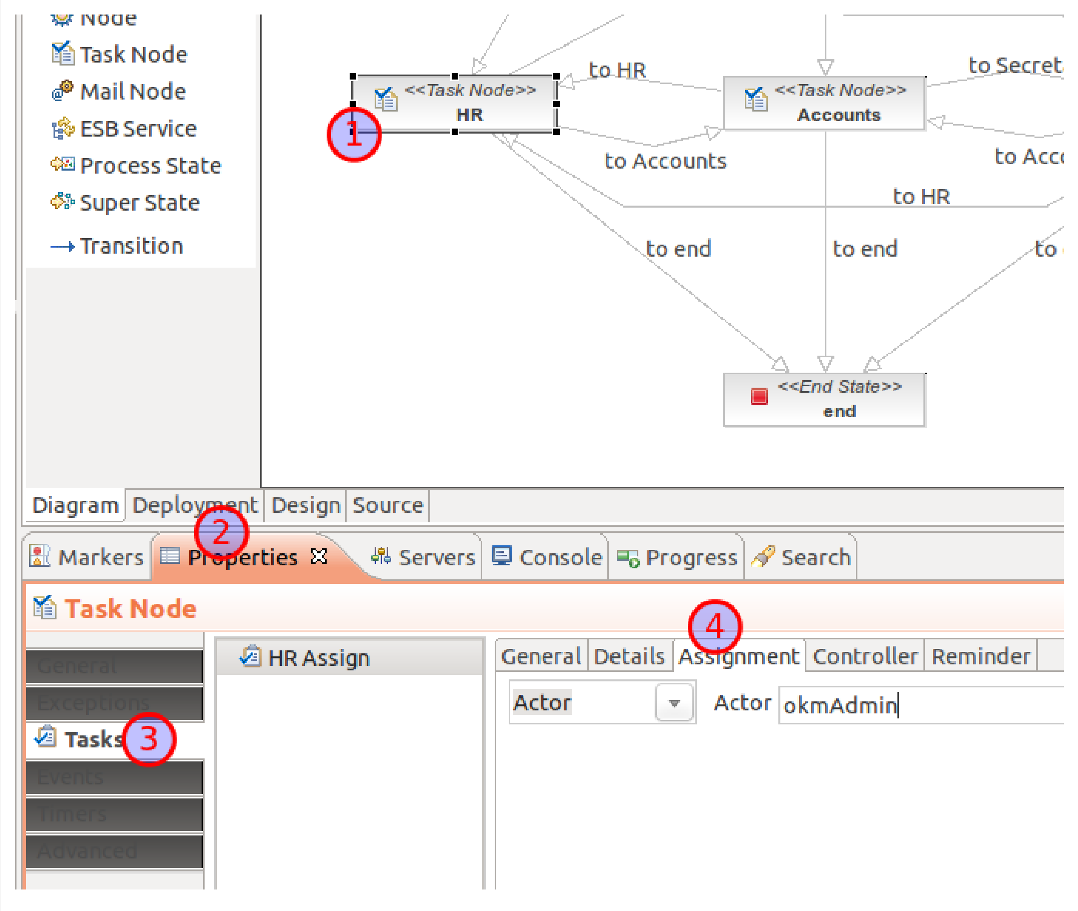
In order to create a task in a task node, go to the properties of this node. In the tasks table click on the right mouse button and select **New Task**. Give a name to this task, for example "something_to_do" and a optional description. In the **Assignment** tab we can see different ways to assign a task:

- **Actor**: Here we put the actorId. This is a String which identify the actor who will perform the task. When the process flow enters in the task node, the task is created and is added to the list of this actor's pending task.
- **Pooled Actor**: A sequence of actorId separated by commas. When this task is created is not assigned to any actor. An actor should get it from the list of pooled tasks. Once the actor get the task, it is added to his own list of pending tasks.
- **Swimlane**: A swimlane defines an assignment which is the same for several tasks in a process.
- **Expression**: This is an assignment expression evaluated by the jBPM identity component. Can make assignment by user or role.
- **Handler**: Here you can specify a class which implements the AssignmentHandler [1] interface. This class assigns a TaskInstance [2] or a SwimlaneInstance [3] to an swimlaneActorId or a set of PooledActors [4].

In this case we are going to assign the task to an user called "yoda":

```xml
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="task node">
    <start-state name="start">
        <transition to="task"></transition>
    </start-state>

    <task-node name="task">
        <task name="something_to_do">
            <assignment actor-id="yoda"></assignment>
        </task>
        <transition to="end"></transition>
    </task-node>

    <end-state name="end"></end-state>
</process-definition>
```

But changing task assignment from the graphical editor is more simple. Let's see this sample workflow with a task node called "HR" which have a task called "HR Assign". These are the steps to change the task assignment:



Actor
Actor okmAdmin

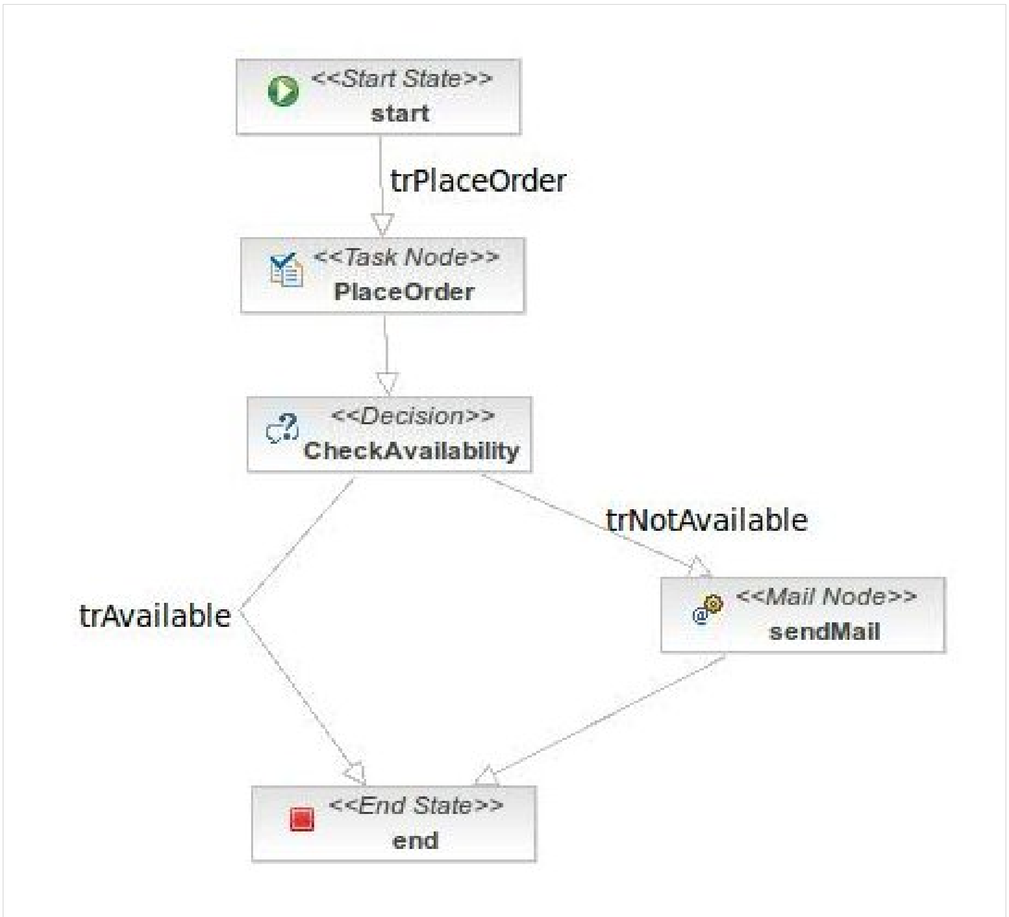
1. Select the task node
2. Click on the properties tab
3. Click on the task section
4. Select the Assignment tab. Here you can select between several ways of assignment. The most simple is the Actor, which is mapped with an OpenKM user.

References

[1] http://docs.jboss.com/jbpm/v3.2/javadoc-jpdl/org/jbpm/taskmgmt/def/AssignmentHandler.html
[2] http://docs.jboss.com/jbpm/v3.2/javadoc-jpdl/org/jbpm/taskmgmt/exe/TaskInstance.html
[3] http://docs.jboss.com/jbpm/v3.2/javadoc-jpdl/org/jbpm/taskmgmt/exe/SwimlaneInstance.html
[4] http://docs.jboss.com/jbpm/v3.2/javadoc-jpdl/org/jbpm/taskmgmt/exe/PooledActor.html

User input request

Now we will create a workflow which need some data from user. This is the workflow graph:



And this is the workflow definition:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<process-definition  xmlns="urn:jbpm.org:jpdl-3.2"  name="UserInput">
  <start-state name="start">
    <transition to="PlaceOrder" name="trPlaceOrder"></transition>
  </start-state>
  <task-node name="PlaceOrder">
    <task name="MyTask">
      <assignment actor-id="tazplat"></assignment>
      <controller></controller>
    </task>
    <transition to="CheckAvailability"></transition>
  </task-node>
  <decision name="CheckAvailability" expression="#{(amount&gt;100)?'trNotAvailable':'trAvailable'}">
```

```
    <transition to="end" name="trAvailable"></transition>
    <transition to="sendMail" name="trNotAvailable"></transition>
  </decision>
  <mail-node name="sendMail" to="stock@your-domain.com">
    <subject>We need more product</subject>
    <text>There is no product, so we must to buy more units!</text>
    <transition to="end"></transition>
  </mail-node>
  <end-state name="end"></end-state>
</process-definition>
```

And this is the form for the user input value (for more info read Workflow Forms description):

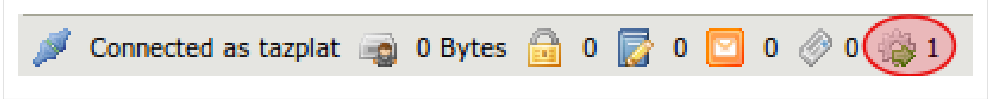
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.0//EN"

"http://www.openkm.com/dtd/workflow-forms-2.0.dtd">
<workflow-forms>
  <workflow-form task="MyTask">
    <input label="Amount" name="amount" />
    <button label="Submit" />
  </workflow-form>
</workflow-forms>
```

This process definition has a task node (with a task called "MyTask") which is assigned to the user "tazplat". You should change this user to meet an existing user in you OpenKM installation. The user have to enter a value for the amount variable. This value is evaluated in the CheckAvailability decision node by this expression:

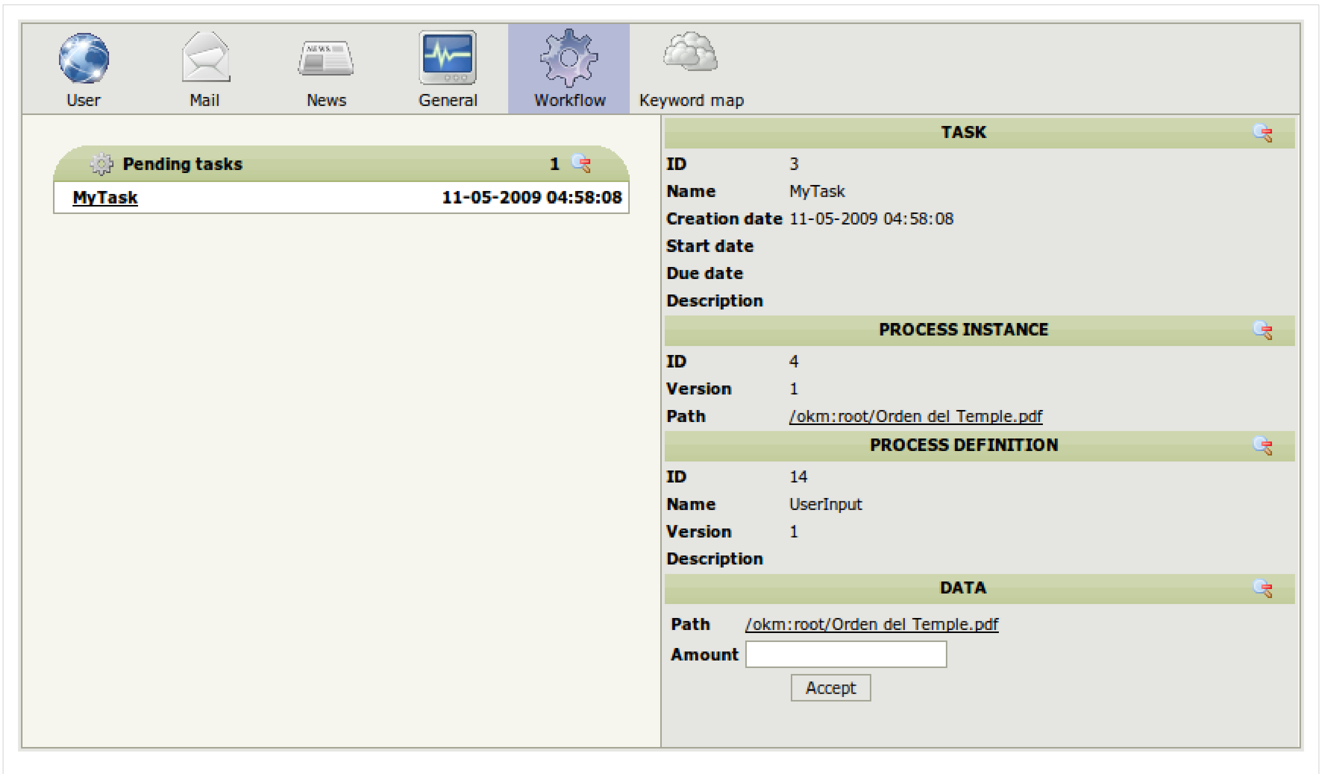
```
#{(amount.value>100)?'trNotAvailable':'trAvailable'}
```

And depending on the requested value, the flow will go across the sendMail mail node or directly to the end state.

Once the process definition is deployed in OpenKM, any user can start the workflow. When the the user input is required, a mail is sent to the user "tazplat". He should log into OpenKM web interface and will see something like this notification:



If he click on this icon, the view will switch to the dashboard. Select the task listed at **Pending** tasks and he will see the details. In the data section are displayed the variables attached to this process instance: an unmodifiable one called **Path** which describe the associated document, and an input called **Amount** where the user can enter a quantity.



Pending tasks 1

MyTask 11-05-2009 04:58:08

TASK

ID 3
Name MyTask
Creation date 11-05-2009 04:58:08
Start date
Due date
Description

PROCESS INSTANCE

ID 4
Version 1
Path /okm:root/Orden del Temple.pdf

PROCESS DEFINITION

ID 14
Name UserInput
Version 1
Description

DATA

Path /okm:root/Orden del Temple.pdf
Amount

Workflow Forms definition

- For **OpenKM 6.2** the workflow forms DTD is **workflow-forms-2.2.dtd**.
- For **OpenKM 6.1** the workflow forms DTD is **workflow-forms-2.2.dtd**.
- For **OpenKM 5.1** the workflow forms DTD is **workflow-forms-2.2.dtd**.
- For **OpenKM 5.0** the workflow forms DTD is **workflow-forms-1.1.dtd**.
- For **OpenKM 4.1** is **workflow-forms-1.0.dtd**.

When you want to retrieve data from an user in the workflow, you have to define a way to do so. OpenKM uses the forms.xml file to draw the components in the form. See this sample:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.2//EN"

"http://www.openkm.com/dtd/workflow-forms-2.2.dtd">
<workflow-forms>
  <workflow-form task="MyTask">
    <input label="Amount" name="amount" />
    <button label="Submit" />
  </workflow-form>
</workflow-forms>
```

Here we can see an XML which describes the components related to a task called "MyTask". This means that when when workflow is executing the "MyTask" task, it need the user who is assigned this task to enter some information to continue. In the description, we can see an INPUT where the user should type an amount.

As you can see in the XML DOCTYPE, there is a formal definition at http://www.openkm.com/dtd/workflow-forms-2.1.dtd.

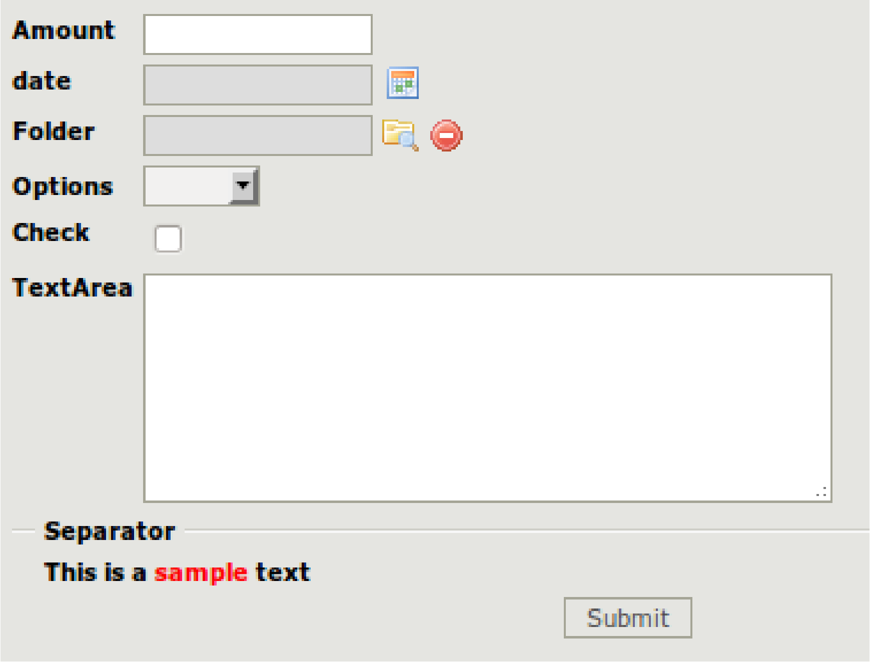
If your OpenKM server can't access to Internet, you can download this DTD and copy to a conveniente place at you server. Remember to update your forms.xml with the TDT location.

```
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.1//EN"

"file:///path/to/workflow-forms-2.1.dtd">
```

Now let's see a more complete form definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.1//EN"

"http://www.openkm.com/dtd/workflow-forms-2.1.dtd">
<workflow-forms>
  <workflow-form task="Sample">
    <input name="input" label="Input"/>
    <input name="date" label="Date" type="date"/>
    <input name="folder" label="Folder" type="folder"></input>
    <select name="options" label="Options">
      <option label="One" value="one"/>
      <option label="Two" value="two"/>
      <option label="Three" value="three"/>
    </select>
    <checkbox name="check" label="Check"/>
    <textarea name="textarea" label="TextArea"/>
    <separator name="separator" label="Separator"/>
    <text name="text" label="This is a &lt;font style='color: red'&gt;sample&lt;/font&gt; text"/>
    <button name="submit" label="Submit" />
  </workflow-form>
</workflow-forms>
```

This form definition is rendered as this:

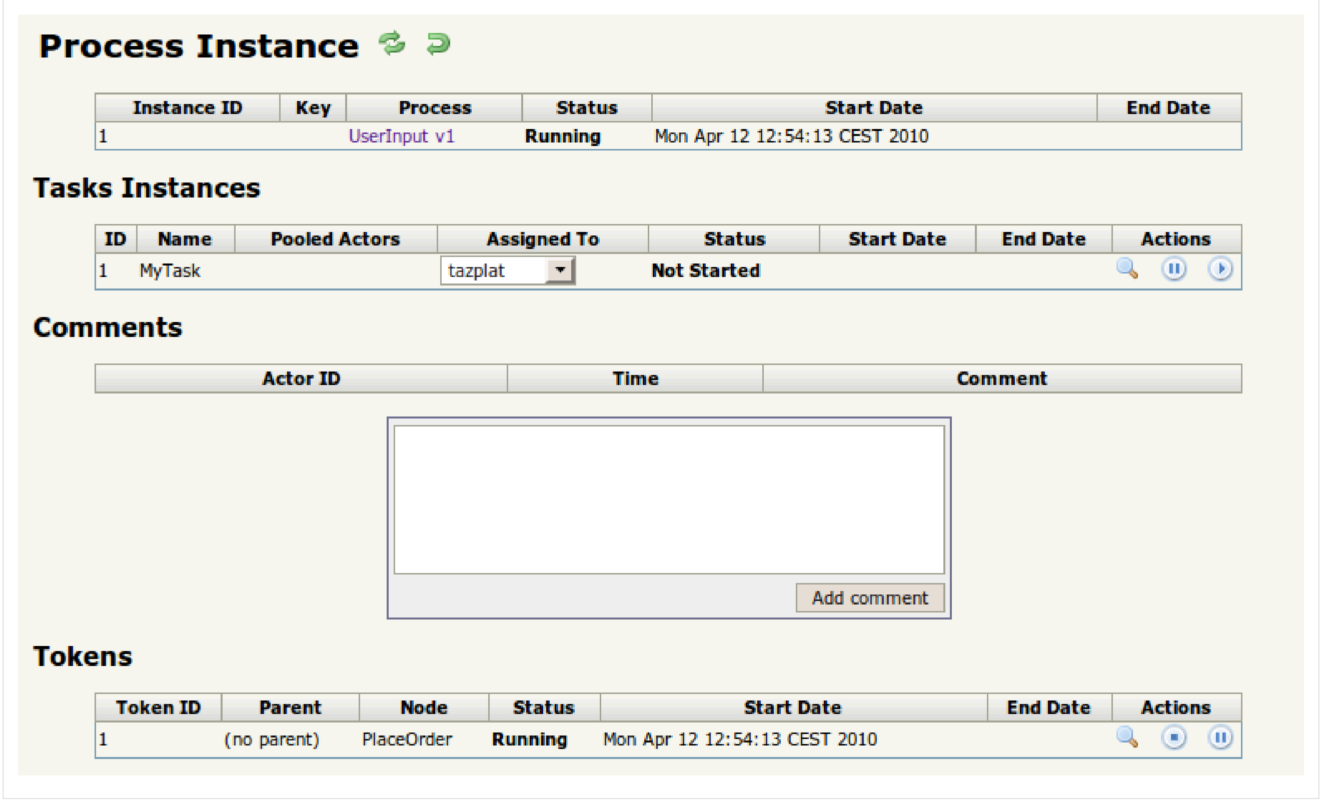


This is a sample text

See Form Element description for a detailed definition of every form element. These form elements have also an additional property called "readonly" with can be used to make a property not modifiable by the user.

Administration interface

If you log as **okmAdmin**, you can access to the administration web interface and will see detailed info about the deployed process definitions and running process instances:



Process Instance

Instance ID	Key	Process	Status	Start Date	End Date
1		UserInput v1	**Running**	Mon Apr 12 12:54:13 CEST 2010	

Tasks Instances

ID	Name	Pooled Actors	Assigned To	Status	Start Date	End Date	Actions
1	MyTask		tazplat	**Not Started**			

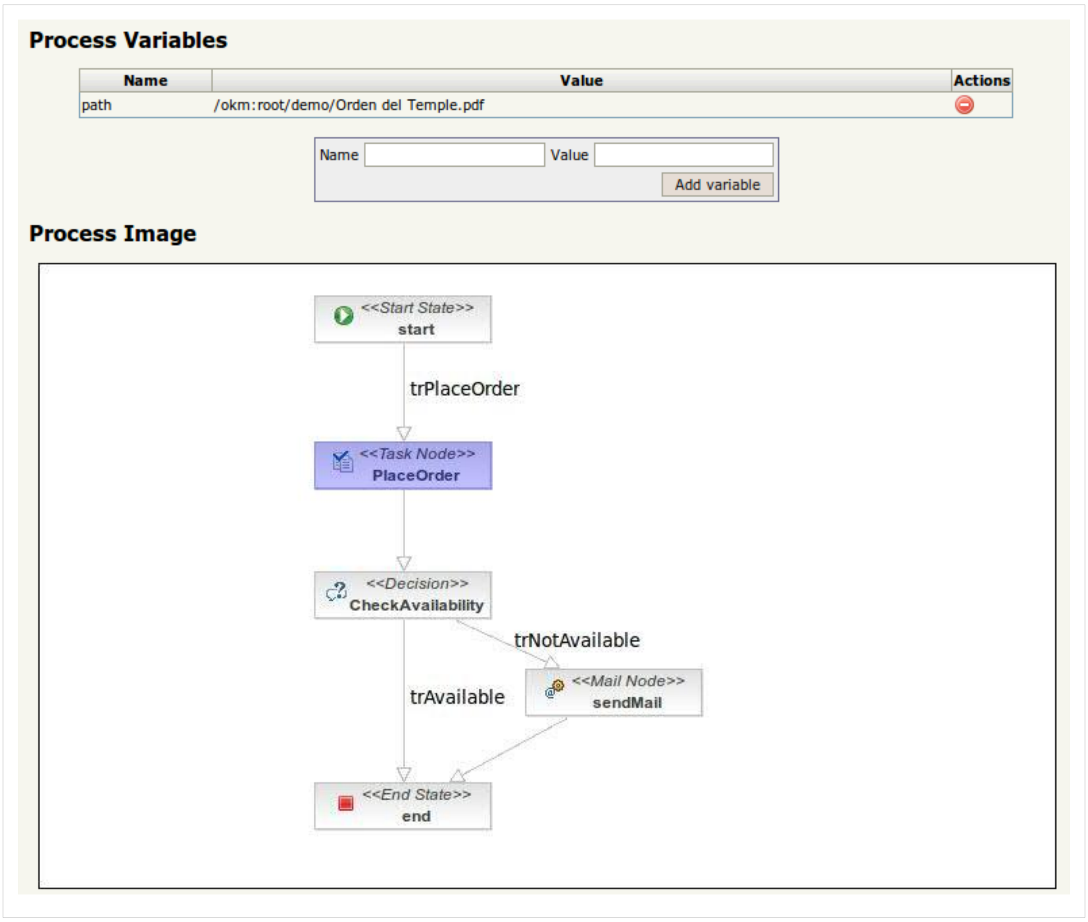
Comments

Actor ID	Time	Comment

Tokens

Token ID	Parent	Node	Status	Start Date	End Date	Actions
1	(no parent)	PlaceOrder	**Running**	Mon Apr 12 12:54:13 CEST 2010		

Also you can see the process definition graph with the current executed node highlighted:



Process Variables

Name	Value	Actions
path	/okm:root/demo/Orden del Temple.pdf	⊖

Name ☐ Value ☐

Process Image

Examples

These are three sample process definitions, each one with the process image, the source xml and the associated handlers (if any). There is also a fourth element called **form definition** which is only available from OpenKM 5.0 and actually is under heavy testing. Please, contact us [1] if you want to test this new feature.

- Examples: Simple 🔒
- Examples: Medium 🔒
- Examples: Advanced 🔒

Also you can see a basic example at:

- Ejemplo Básico jBPM (I) [2] (spanish)
- Ejemplo Básico jBPM (y II) [3] (spanish)

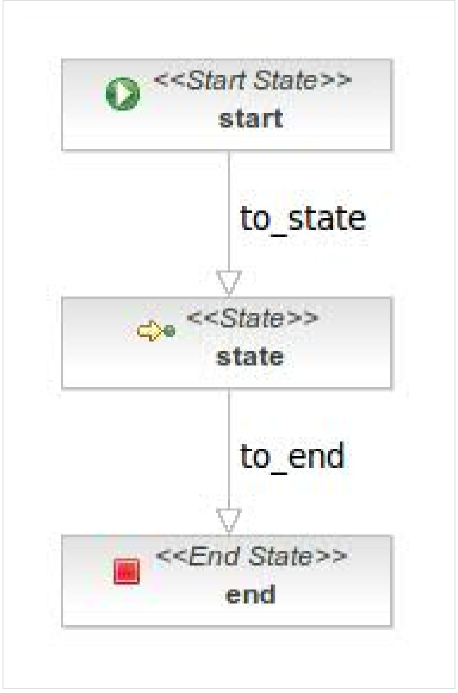
References

[1] http://www.openkm.com/Contact
[2] http://www.workflowsworld.com/2009/01/ejemplo-basico-jbpm-i/
[3] http://www.workflowsworld.com/2009/01/ejemplo-basico-jbpm-y-ii/

Examples: Simple

Download and test this process definition: .

Process image



Process definition

```xml
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="simple">
  <start-state name="start">
    <transition name="to_state" to="state"></transition>
    <event type="node-leave">
      <script>
        print(&quot;Node start&quot;);
      </script>
    </event>
  </start-state>
  <state name="state">
    <event type="node-enter">
      <script>
        print(&quot;Node state&quot;);
        executionContext.leaveNode();
      </script>
    </event>
    <transition name="to_end" to="end">
      <action name="action" class="com.openkm.sample.MessageActionHandler">
        <message>About to finish!</message>
      </action>
    </transition>
  </state>
  <end-state name="end">
    <event type="node-enter">
      <script>
        print(&quot;Node end
(&quot;+executionContext.getVariable(&quot;message&quot;)+&quot;)&quot;);
      </script>
    </event>
  </end-state>
</process-definition>
```

Process handlers

```java
package com.openkm;

import org.jbpm.graph.def.ActionHandler;
import org.jbpm.graph.exe.ExecutionContext;

public class MessageActionHandler implements ActionHandler {
  private static final long serialVersionUID = 1L;

  /**
   * The message member gets its value from the configuration in the
   * processdefinition. The value is injected directly by the engine.
   */
  String message;

  /**
   * A message process variable is assigned the value of the message
   * member. The process variable is created if it doesn't exist yet.
   */
  @Override
  public void execute(ExecutionContext context) throws Exception {
    context.getContextInstance().setVariable("message", message);
    System.out.println("From MessageActionHandler...");
  }
}
```

Form definition

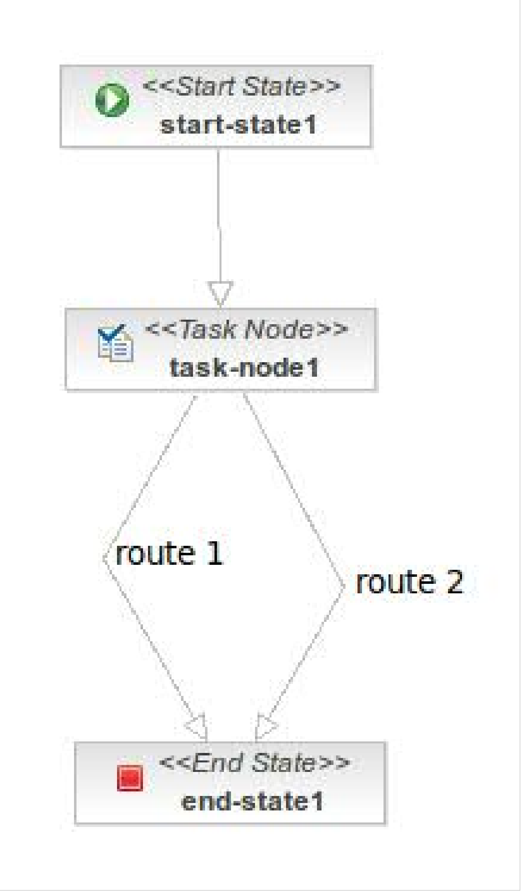
None

Examples: Medium

Download and test this process definition: .
If you see log messages like these, don't worry because are normal:

```
INFO  [JpdlXmlReader] process xml information: no swimlane or assignment specified for task
'<task xmlns="urn:jbpm.org:jpdl-3.2" name="start" blocking="false" signalling="true" priority="normal" notify="false">
  <description>Task sample</description>
</task>'
WARN  [ProxyWarnLog] Narrowing proxy to class org.jbpm.graph.node.StartState - this operation breaks ==
WARN  [ProxyWarnLog] Narrowing proxy to class org.jbpm.graph.node.TaskNode - this operation breaks ==
WARN  [ProxyWarnLog] Narrowing proxy to class org.jbpm.graph.node.TaskNode - this operation breaks ==
```

Process image



Process definition

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="medium">
  <start-state name="start-state1">
    <task name="start">
      <description>Task sample</description>
    </task>
    <transition to="task-node1"></transition>
  </start-state>

  <task-node name="task-node1">
    <task name="user_info" priority="low">
      <assignment actor-id="okmAdmin"></assignment>
      <event type="task-create">
        <script>taskInstance.start();</script>
      </event>
    </task>
    <transition to="end-state1" name="route 1">
      <script>print(&quot;Going through: route 1&quot;);</script>
    </transition>
    <transition to="end-state1" name="route 2">
```

```
      <script>print(&quot;Going through: route 2&quot;);</script>
    </transition>
  </task-node>

  <end-state name="end-state1">
    <event type="node-enter">
      <script name="mensajes">
        print(&quot;End node reached: &quot;+node);
        print(&quot;Var 'quantity':
&quot;+executionContext.getVariable(&quot;quantity&quot;));
        print(&quot;Var 'name':
&quot;+executionContext.getVariable(&quot;name&quot;));
        print(&quot;Var 'type':
&quot;+executionContext.getVariable(&quot;type&quot;));
      </script>
    </event>
  </end-state>
</process-definition>
```

Process handlers

None

Form definition

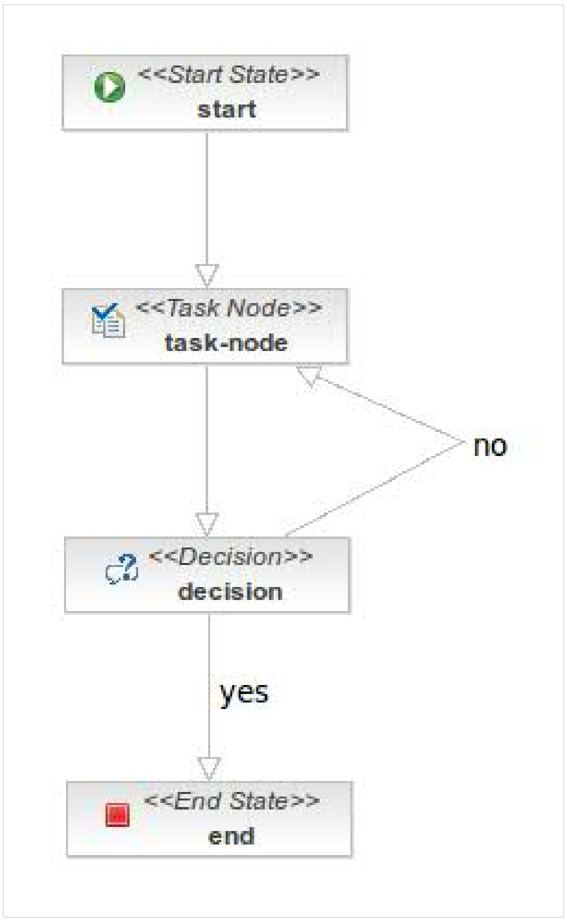
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 1.1//EN"

"http://www.openkm.com/dtd/workflow-forms-1.1.dtd">
<workflow-forms>
  <workflow-form task="start">
        <input label="Quantity" name="quantity" value="10"/>
        <button name="save" label="Save" />
  </workflow-form>
  <workflow-form task="user_info">
    <input label="Name" type="text" name="name" value="John" />
    <input label="Surname" type="text" name="surname" value="Doe" />
    <textarea label="Info" name="info" value=""/>
    <select label="Type" name="type" type="simple">
      <option label="Type 1" value="t1" />
      <option label="Type 2" value="t2" selected="true" />
      <option label="Type 3" value="t3" />
    </select>
    <button name="goto1" label="Goto 1" value="route 1" type="transition" />
    <button name="goto2" label="Goto 2" value="route 2" type="transition" />
  </workflow-form>
</workflow-forms>
```

Examples: Advanced

Download and test this process definition: .

The taks will be assigned to an user called "monkiki" so you need to create this user and log as him to see the task assignment. Also you can assing this task to another user from the process instance workflow administration.

Process image



Process definition

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition  xmlns="urn:jbpm.org:jpdl-3.2"  name="advanced">
  <start-state name="start">
    <transition to="task-node"></transition>
  </start-state>

  <task-node name="task-node">
    <task name="guess_a_number">
```

```
      <assignment actor-id="monkiki"></assignment>
      <event type="task-create">
        <script>taskInstance.start();</script>
      </event>
    </task>
    <transition to="decision"></transition>
  </task-node>

  <decision name="decision">
    <handler class="com.openkm.sample.VerifyNumber"></handler>
      <transition to="task-node" name="no"></transition>
      <transition to="end" name="yes"></transition>
    </decision>

  <end-state name="end"></end-state>
</process-definition>
```

Process handlers

```
package com.openkm.sample;

import com.openkm.bean.form.Input;

import org.jbpm.graph.exe.Comment;
import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.graph.node.DecisionHandler;

public class VerifyNumber implements DecisionHandler {
  private static final long serialVersionUID = 1L;

  @Override
  public String decide(ExecutionContext executionContext) throws
Exception {
    Input numberStr = (Input)
executionContext.getContextInstance().getVariable("number");
    Input guessStr = (Input)
executionContext.getContextInstance().getVariable("guess");
    System.out.println("numberStr: " + numberStr);
    System.out.println("guessStr: " + numberStr);

    Integer guess = Integer.valueOf(guessStr.getValue());
    Integer number;

    if (numberStr != null && !numberStr.equals("")) {
      number = Integer.valueOf(numberStr.getValue());
    } else {
      number = 10;
    }
    if (guess > number) {
      System.out.println("Too high!");
      executionContext.getToken().addComment(new Comment("system",
guess + " is too high!"));
      return "no";
    } else if (guess < number) {
      System.out.println("Too low!");
      executionContext.getToken().addComment(new Comment("system",
guess + " is too low!"));
      return "no";
    } else {
      System.out.println("Great!");
      executionContext.getToken().addComment(new Comment("system",
guess + " is great!"));
      return "yes";
    }
  }
}
```

Form definition

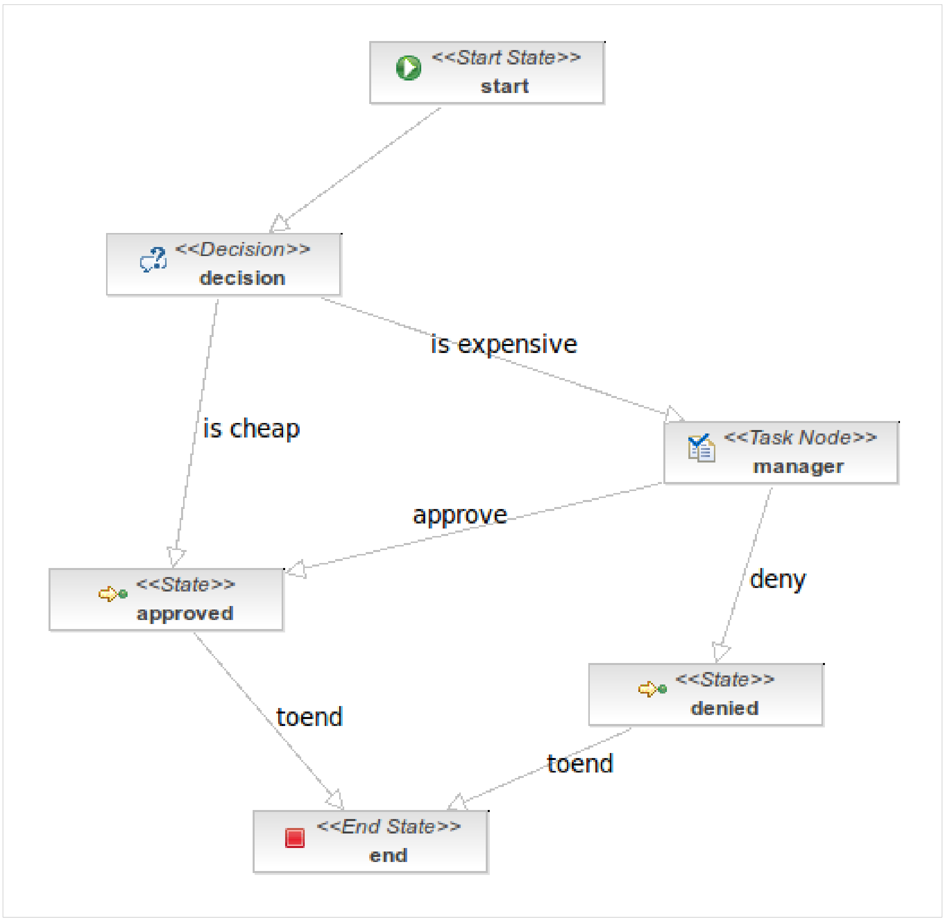
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.1//EN"

"http://www.openkm.com/dtd/workflow-forms-2.1.dtd">
<workflow-forms>
  <workflow-form task="run_config">
    <input label="Number to guess" name="number" />
    <button name="submit" label="Submit" />
  </workflow-form>
  <workflow-form task="guess_a_number">
    <input label="Guess" name="guess" />
    <button name="submit" label="Submit" />
  </workflow-form>
</workflow-forms>
```

Examples: Purchase

Download and test this process definition: .

The taks will be assigned to an user called "manager" so you need to create this user and log as him to see the task assignment. Also you can assing this task to another user from the process instance workflow administration.

Process image



Process definition

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="purchase">
  <start-state name="start">
    <transition to="decision"></transition>
  </start-state>

  <decision name="decision">
    <transition to="approved" name="is cheap">
```

```
      <condition expression="#{price.value &lt;= 500}"></condition>
    </transition>
    <transition to="manager" name="is expensive">
      <condition expression="#{price.value &gt; 500}"></condition>
    </transition>
  </decision>

  <task-node name="manager">
    <task name="evaluate price">
      <description>The manager may deny purchase or go ahead.</description>
      <assignment actor-id="manager"></assignment>
    </task>
    <transition to="denied" name="deny"></transition>
    <transition to="approved" name="approve"></transition>
  </task-node>

  <state name="approved">
    <description>The purchase has been approved.</description>
    <timer duedate="15 seconds" name="approved timer" transition="toend">
      <script>print(&quot;From APPROVED Go to END&quot;);</script>
    </timer>
    <transition to="end" name="toend"></transition>
  </state>

  <state name="denied">
    <description>The purchase has been denied.</description>
    <timer duedate="15 seconds" name="denied timer">
      <script>print(&quot;From DENIED Go to END&quot;);</script>
    </timer>
    <transition to="end" name="toend"></transition>
  </state>

  <end-state name="end"></end-state>
</process-definition>
```

Process handlers

None.

Form definition

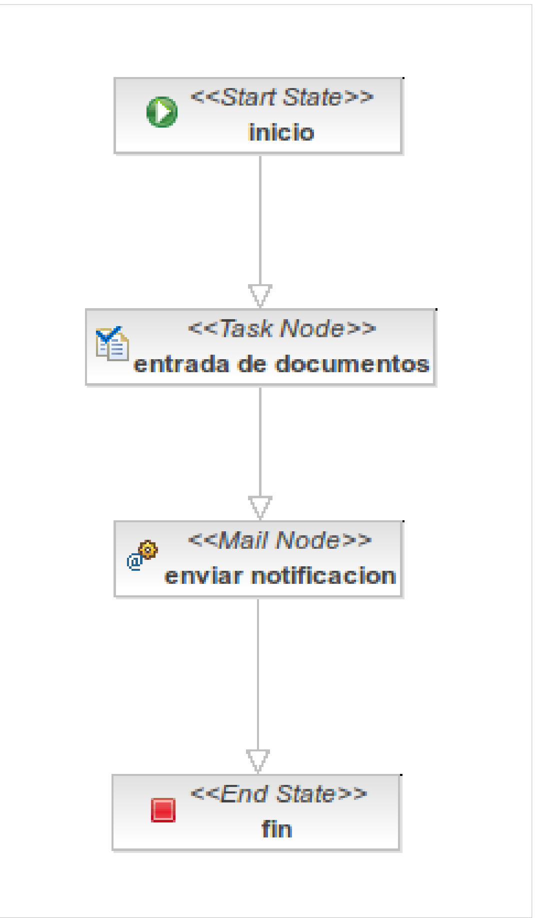
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.0//EN"

"http://www.openkm.com/dtd/workflow-forms-2.0.dtd">
<workflow-forms>
  <workflow-form task="run_config">
    <input label="Purchase price" name="price" />
    <textarea label="Purchase description" name="description" />
    <button name="submit" label="Submit" />
  </workflow-form>
  <workflow-form task="evaluate price">
    <input label="Purchase price" name="price" data="price" readonly="true" />
    <textarea label="Purchase description" name="description" data="description" readonly="true" />
    <button name="approve" label="Approve" transition="approve"/>
    <button name="deny" label="Deny" transition="deny"/>
  </workflow-form>
</workflow-forms>
```

Examples: Process Handler

Download and test this process definition: .

The taks will be assigned to an user called "openkm" so you need to create this user and log as him to see the task assignment. Also you can assing this task to another user from the process instance workflow administration.

Process image



Process definition

```xml
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="phandler">
   <start-state name="inicio">
      <transition to="entrada de documentos">
         <action class="com.openkm.sample.FechaHandler"></action>
      </transition>
   </start-state>

   <task-node name="entrada de documentos">
      <task name="Recepcion de Documentos">
         <assignment actor-id="openkm"></assignment>
      </task>
      <event type="node-enter">
         <script>
            print(&quot; ent_fecha &quot;)
         </script>
      </event>
      <transition to="enviar notificacion">
         <action class=""></action>
      </transition>
   </task-node>

   <mail-node name="enviar notificacion">
      <transition to="fin"></transition>
   </mail-node>

   <end-state name="fin"></end-state>
</process-definition>
```

Process handlers

```
package com.openkm.sample;

import java.util.Calendar;

import org.jbpm.graph.def.ActionHandler;
import org.jbpm.graph.exe.ExecutionContext;

import com.openkm.bean.form.Input;
import com.openkm.util.ISO8601;

public class FechaHandler implements ActionHandler {
    private static final long serialVersionUID = 1L;

    @Override
    public void execute(ExecutionContext context) throws Exception {
        Input myFecha = new Input();
        myFecha.setName("ent_fecha");
        myFecha.setValue(ISO8601.format(Calendar.getInstance()));
        context.getContextInstance().setVariable("ent_fecha", myFecha);
    }
}
```

Form definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow-forms PUBLIC "-//OpenKM//DTD Workflow Forms 2.0//EN"

"http://www.openkm.com/dtd/workflow-forms-2.0.dtd">
<workflow-forms>
  <workflow-form task="Recepcion de Documentos">
    <separator label="REGISTRO DE ENTRADA" name="sep_grp_entrada"/>

    <input label="No." name="ent_id"/>
    <input label="Fecha" name="ent_fecha" type="date" data="ent_fecha" readonly="true"/>
    <select label="Medio" name="ent_medio" type="simple">
      <option label="Correo" value="correo" selected="true"/>
      <option label="Email" value="email" />
      <option label="Fax" value="fax" />
      <option label="Mensajeria" value="mensajeria" />
      <option label="Propia Mano" value="propia mano" />
    </select>

    <separator label="DOCUMENTO" name="sep_doc"/>

    <input label="No." name="ent_no_doc"/>
    <input label="Fecha" name="ent_fecha_doc" type="date"/>
    <textarea label="Asunto" name="ent_asunto" value=""/>

    <separator label="OBSERVACIONES" name="sep_obs"/>

    <textarea label="Observaciones" name="ent_obs" value=""/>
    <input label="Capturo" name="ent_capturo"/>
    <button label="Aceptar" name="ent_aceptar"/>
  </workflow-form>
</workflow-forms>
```

Article Sources and Contributors

Workflow Guide *Source*: http://wiki.openkm.com/index.php?oldid=6026 *Contributors*: Pavila

Overview *Source*: http://wiki.openkm.com/index.php?oldid=1583 *Contributors*: Pavila

JBPM installation *Source*: http://wiki.openkm.com/index.php?oldid=5921 *Contributors*: Pavila

JBPM configuration *Source*: http://wiki.openkm.com/index.php?oldid=7531 *Contributors*: Anonymous, Jllort, Pavila

Eclipse plugin *Source*: http://wiki.openkm.com/index.php?oldid=870 *Contributors*: Pavila

Eclipse plugin: Installation *Source*: http://wiki.openkm.com/index.php?oldid=4591 *Contributors*: Pavila

Eclipse plugin: Usage *Source*: http://wiki.openkm.com/index.php?oldid=6164 *Contributors*: Pavila

Process modelling *Source*: http://wiki.openkm.com/index.php?oldid=6317 *Contributors*: Pavila

Hello world! *Source*: http://wiki.openkm.com/index.php?oldid=1015 *Contributors*: Pavila

Starting a workflow *Source*: http://wiki.openkm.com/index.php?oldid=6967 *Contributors*: Pavila

Use of node node *Source*: http://wiki.openkm.com/index.php?oldid=6060 *Contributors*: Pavila

Use of decision node *Source*: http://wiki.openkm.com/index.php?oldid=6062 *Contributors*: Pavila

Use of task node *Source*: http://wiki.openkm.com/index.php?oldid=6143 *Contributors*: Pavila

User input request *Source*: http://wiki.openkm.com/index.php?oldid=4651 *Contributors*: Pavila

Workflow Forms definition *Source*: http://wiki.openkm.com/index.php?oldid=7402 *Contributors*: Pavila

Administration interface *Source*: http://wiki.openkm.com/index.php?oldid=1010 *Contributors*: Pavila

Examples *Source*: http://wiki.openkm.com/index.php?oldid=4468 *Contributors*: Pavila

Examples: Simple *Source*: http://wiki.openkm.com/index.php?oldid=4485 *Contributors*: Pavila

Examples: Medium *Source*: http://wiki.openkm.com/index.php?oldid=6209 *Contributors*: Pavila, Suresh.grandhi.npb

Examples: Advanced *Source*: http://wiki.openkm.com/index.php?oldid=6165 *Contributors*: Pavila

Examples: Purchase *Source*: http://wiki.openkm.com/index.php?oldid=4704 *Contributors*: Pavila

Examples: Process Handler *Source*: http://wiki.openkm.com/index.php?oldid=4971 *Contributors*: Pavila

Image Sources, Licenses and Contributors

File:Padlock.gif *Source*: http://wiki.openkm.com/index.php?title=File:Padlock.gif *License*: unknown *Contributors*: Pavila

File:Nota idea.png *Source*: http://wiki.openkm.com/index.php?title=File:Nota_idea.png *License*: unknown *Contributors*: Pavila

File:Okm workflow guide 001.png *Source*: http://wiki.openkm.com/index.php?title=File:Okm_workflow_guide_001.png *License*: unknown *Contributors*: Pavila

File:Jbpm install 01.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_install_01.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm install 02.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_install_02.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm install 03.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_install_03.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm install 04.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_install_04.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm install 05.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_install_05.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm install 06.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_install_06.jpg *License*: unknown *Contributors*: Pavila

File:Nota clasica.png *Source*: http://wiki.openkm.com/index.php?title=File:Nota_clasica.png *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse install 01.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_install_01.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse install 02.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_install_02.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse install 03.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_install_03.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 01.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_01.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 02.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_02.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 03.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_03.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 04.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_04.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 05.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_05.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 06.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_06.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 07.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_07.jpg *License*: unknown *Contributors*: Pavila

File:Jbpm eclipse new 08.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_eclipse_new_08.jpg *License*: unknown *Contributors*: Pavila

File:Workflow_example_simple.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_example_simple.png *License*: unknown *Contributors*: Pavila

File:Jbpm node start.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_start.png *License*: unknown *Contributors*: Pavila

File:Jbpm node end.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_end.png *License*: unknown *Contributors*: Pavila

File:Jbpm node task.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_task.png *License*: unknown *Contributors*: Pavila

File:Jbpm node node.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_node.png *License*: unknown *Contributors*: Pavila

File:Jbpm node state.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_state.png *License*: unknown *Contributors*: Pavila

File:Jbpm node decision.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_decision.png *License*: unknown *Contributors*: Pavila

File:Jbpm node fork.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_fork.png *License*: unknown *Contributors*: Pavila

File:Jbpm node join.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_join.png *License*: unknown *Contributors*: Pavila

File:Hello.world.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Hello.world.jpg *License*: unknown *Contributors*: Pavila

File:Openkm.toolbar.workflow.png *Source*: http://wiki.openkm.com/index.php?title=File:Openkm.toolbar.workflow.png *License*: unknown *Contributors*: Pavila

File:Nota_advertencia.png *Source*: http://wiki.openkm.com/index.php?title=File:Nota_advertencia.png *License*: unknown *Contributors*: Pavila

File:Workflow samples.zip *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_samples.zip *License*: unknown *Contributors*: Pavila

File:Jbpm sample node.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_sample_node.png *License*: unknown *Contributors*: Pavila

File:Jbpm sample decision.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_sample_decision.png *License*: unknown *Contributors*: Pavila

File:Jbpm sample task.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_sample_task.png *License*: unknown *Contributors*: Pavila

File:Jbpm node task assignment.png *Source*: http://wiki.openkm.com/index.php?title=File:Jbpm_node_task_assignment.png *License*: unknown *Contributors*: Pavila

File:Workflow.user.input.jpg *Source*: http://wiki.openkm.com/index.php?title=File:Workflow.user.input.jpg *License*: unknown *Contributors*: Pavila

File:Openkm.status.workflow.png *Source*: http://wiki.openkm.com/index.php?title=File:Openkm.status.workflow.png *License*: unknown *Contributors*: Pavila

File:Openkm.dashboard.workflow.png *Source*: http://wiki.openkm.com/index.php?title=File:Openkm.dashboard.workflow.png *License*: unknown *Contributors*: Pavila

File:Workflow Forms 1.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_Forms_1.png *License*: unknown *Contributors*: Pavila

File:Openkm.admin.workflow.1.png *Source*: http://wiki.openkm.com/index.php?title=File:Openkm.admin.workflow.1.png *License*: unknown *Contributors*: Pavila

File:Openkm.admin.workflow.2.png *Source*: http://wiki.openkm.com/index.php?title=File:Openkm.admin.workflow.2.png *License*: unknown *Contributors*: Pavila

File:Simple.par *Source*: http://wiki.openkm.com/index.php?title=File:Simple.par *License*: unknown *Contributors*: Pavila

File:Workflow example simple.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_example_simple.png *License*: unknown *Contributors*: Pavila

File:Medium.par *Source*: http://wiki.openkm.com/index.php?title=File:Medium.par *License*: unknown *Contributors*: Pavila

File:Workflow example medium.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_example_medium.png *License*: unknown *Contributors*: Pavila

File:Advanced.par *Source*: http://wiki.openkm.com/index.php?title=File:Advanced.par *License*: unknown *Contributors*: Pavila

File:Workflow example advanced.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_example_advanced.png *License*: unknown *Contributors*: Pavila

File:Purchase.par *Source*: http://wiki.openkm.com/index.php?title=File:Purchase.par *License*: unknown *Contributors*: Pavila

File:Workflow example purchase.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_example_purchase.png *License*: unknown *Contributors*: Pavila

File:Phandler.par *Source*: http://wiki.openkm.com/index.php?title=File:Phandler.par *License*: unknown *Contributors*: Pavila

File:Workflow example process handler.png *Source*: http://wiki.openkm.com/index.php?title=File:Workflow_example_process_handler.png *License*: unknown *Contributors*: Pavila