Chapter No. 4
"The Prototype user interface"
In this package, you will find:

A Biography of the author of the book
A preview chapter from the book, Chapter No. 4 “A Prototype User Interface"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Matt Cumberlidge is a business analyst working for a world leading FTSE 100 provider of information-driven services and solutions based in Oxford, UK. In this role, Matt has undertaken a very wide range of projects, but the common theme running throughout is that of business process. Over the last year or so Matt has extended his core capabilities in business process analysis and re-engineering into the realm of business process management and in particular an investigation of the JBoss jBPM implementation. Matt is delighted to be able to share his experiences and ideas about this exciting technology with a wider audience through the publication of this book.


This book shows business analysts how to model business processes in JBoss jBPM and use these models to generate a fully-functioning workflow application. It shows how business analysts can use the tools to build a solution without the need for Java coding expertise. It also introduces more advanced functionality that can be implemented by Java developers in partnership with the Business Analyst.

This book takes a practical approach, with step-by-step instructions for business process management, model creation, and implementation. It uses a typical BPM project lifecycle case study to explore and explain the process in a realistic situation.

What This Book Covers

Chapter 1 discusses the background from which BPM has emerged, and how BPM fits into the wider scheme of enterprise application development. We define what BPM means for us, and look at the business scenarios where BPM is the right solution. Also, we introduce our suggested BPM project lifecycle, and see the tools that we'll put together as our open source-based BPM suite.

Chapter 2 covers all the major tools in the process analyst's kit bag, with a view to creating a deep understanding of the process we are seeking to systematize in our BPMS.

Chapter 3 covers the software installations—Java, the JBoss application server, the jBPM engine, and the jBPM Designer. Also, we take a look at the fundamental concepts that underpin JBoss jBPM and put these concepts into practice by building our first process definition for our proof-of-concept system.

Chapter 4 covers building the user interface that our proof-of-concept testers will use to interact with the process definition that we built in the previous chapter.

Chapter 5 covers putting the jBPM system on a server so our proof-of-concept testers can bash their test data into it and give us feedback on what they think. Also, how we can allow managers to prioritize tasks by design and on the fly. Most complicated of all, we see how our system can be integrated with other applications, both in house and external.

Chapter 6 looks at how we judge when we are ready to start planning to go live and also covers the essentials we need to consider when building an implementation plan. We show how the web console can be customized according to your own branding and we see how we can swap the default jBPM database for a more robust, enterprise-ready database server. We will also integrate and put to use the SeeWhy Business Activity Monitoring solution.

Chapter 7 covers how to assess our project and perform process analysis and ongoing improvement. We also put together business process documentation, and present ideas for further development of our BPM system.

It's all very well having a beautiful process definition, but this is "business process management", not "business process definition". Having a process definition that accurately reflects the way the business works is only half the story. In this chapter, we'll add the majority of the remaining half. The final icing on the cake will be added in the last few chapters.

In this chapter, we are going to build the end user part of our BPM system. We will put together the user interface, which our proof-of-concept testers will use to interact with the process definition that we created in the previous chapter. By the end of the chapter, we will have obtained sign-off from our sponsors and the proof-of-concept testers indicating that they are happy that this user interface is ready to test. In the next chapter, we'll run our testing to prove that our BPM concept system will indeed meet Bland Records' requirements and should be further developed.

In this chapter, we will look at the following:

- Building web forms for the tasks we defined
- Setting up our proof-of-concept users in the system
- Deploying the process
- The default jBPM web console interface
- Adding some help text to the web console interface
- Obtaining sign-off that the web console is ready to run the proof of concept

Build the prototype

With the process engine of our BPM system now in place, it's time to turn our attention to the user interface that our users will interact with, as they execute the process. We will make this available over the internet as a series of web screens that the user can use like a regular website: the jBPM project refers to this as the "web console". For the time being, we will build the web console on our local machine, where we'll be able to play around with it until we are happy.

The prototype user interface

In the next chapter, we'll see how we can deploy it on a server, so our proof-of-concept users can test the system properly.

The web console is a combination of task lists, which represent the queue of work that a user or a group of users has to do, and task forms, which are the screens where the user can actually complete the tasks that have been assigned to them. As we'll see later in the chapter, the jBPM web console also comes with some other handy utilities out of the box, although most of these are more useful for management and administrators than end users.

Develop the prototype user interface

The first step is to build what jBPM terms the "task forms". These are the screens where the user will do the work assigned to them, making an update to a process variable or confirming they have done a piece of offline work, for example:

![Task Instance View]

We need to generate one of these task forms for every task node in our process. This is quite a simple thing to do but, as there isn't a "generate all task forms" button, it is rather time consuming. Still, it does give us the opportunity to ensure each task form is correctly put together. The task forms use XHTML JavaServer Faces tags. JavaServer Faces (JSF) is a Java-based web application framework that simplifies the development of user interfaces for Java applications. For those who know some HTML, the syntax for the Faces tags is not a million miles away. For our purposes, we won't need to worry too much about the Faces elements, as we will limit our interaction with them to a few simple amendments and additions. Nevertheless, for those who are interested, there is a good tutorial on JavaServer Faces here:

http://www.exadel.com/tutorial/jsf/jsftutorial-kickstart.html

Anyway, on with our work. Start up the process Designer if you haven't already got it open. Highlight the first task node in the process diagram, which is of course our start node, "Hold auditions". In the Outline view, expand the tree until you find the "Hold auditions" task (note you need the task element within the node, not the node itself). Right-click and select Properties. In the left-hand menu, navigate to the Advanced screen. On here, you will see a Generate Form... button; click it:
Once you've clicked to create the task form, the Designer will ask you to confirm the variables that you've already defined in the process definition and which should be collected on this task form. In our case, these are `audDate` and `audLocation` for the "Hold auditions" task node:

![Generate Task Form dialog](image)

As mentioned in the previous chapter, it will be the mapped names of "Audition date" and "Audition location" that will actually be used in the web console, making for a more user-friendly experience.

You will notice at the bottom of this dialog window that the Designer automatically gives the task form file a name of `task node name.xhtml`, with `.xhtml` being the filename extension used for XHTML JavaServer Faces files. We should leave these settings as they are: the only time we'd need to change the filename is if we have two nodes called the same thing. Click OK, then OK again, to close the task node properties window and you will see in the Package Explorer window that the new task form file has been added to our project, along with an XML file called `forms.xml`.
Double-click the **Hold-auditions.xhtml** file, so we can have a look at what's been created for us. Something like the following code should open up in the main editor window:

```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!-- the DOCTYPE means we are required to use html for a root element -->
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:ui="http://java.sun.com/jsf/facelets"
     xmlns:c="http://java.sun.com/jstl/core"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:f="http://java.sun.com/jsf/core"
     xmlns:tf="http://jbpm.org/jsf/tf"
     xmlns:jbpm="http://jbpm.org/jsf">

<ui:component>

<jbpm:dataform>

<f:facet name="header">
   <h:outputText value="#{taskName}"/>
</f:facet>

<!-- TASKFORM ROWS -->
```
As you can see, this code really isn't massively different from HTML: it is certainly pretty understandable to the uninitiated. Still, we don't actually have to worry about its content in any respect: we can quite happily leave it completely alone. Nevertheless, a bit later in this chapter, we will make a couple of edits to it, just to tailor it to our needs somewhat.

While we're here, we may as well have a quick look at the forms.xml file: double-click it to inspect it in the main editor window. The content in the Source view will be something like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<forms>
  <form task="Hold auditions" form="Hold-auditions.xhtml"/>
</forms>
```

As you can see, this file is extremely simple and all it does is list out all the task forms that will be needed by the web console to execute the process definition. We will not need to edit this file at all.

Now comes the laborious bit: we must go into every task node that we have defined and repeat the above task form creation process. There should be no need to change any of the variables or file names, just accept the defaults that the Designer gives you. Obviously, you don't create any task forms for fork or join nodes, only for task nodes. When we've finished, we should have a long list of XHTML task form files in our Package Explorer:

If you look at the XHTML code for a node where we have specified a name for a transition from the node, there will be a task form button tag included that will be called the same name as the transition. For example, in our `Contract-supporting-musicians.xhtml` task form, we have the following buttons specified:

```xml
<!-- TASKFORM BUTTONS -->
<tf:saveButton value="Save"/>
<tf:cancelButton value="Cancel"/>
<tf:transitionButton transition="Done" value="Done"/>
```

The prototype user interface

The transitionButton is given a value of "Done", the same name as the transition we specified, and will result in a button labeled "Done" showing up in the web console for our users to click. So even though the user won't actually input any process variables on this screen, they will move the process on by clicking the appropriately-labeled "Done" button to confirm they have done the task offline and the process can resume. Of course, this gets even more useful where we have specified two transitions and the task form will therefore include two buttons for the user to choose from. For example, the user will be able to choose between an "Incorrect" and a "Correct" button on the "Review credits and cover artwork" task form, because the Designer has automatically recognized that there are two leaving transitions and created the necessary buttons with the following code:

```xml
<tf:saveButton value="Save"/>
<tf:cancelButton value="Cancel"/>
<tf:transitionButton transition="Incorrect" value="Incorrect"/>
<tf:transitionButton transition="Correct" value="Correct"/>
```

This is very clever stuff and goes to prove just how much jBPM is doing for us without us having to get our hands dirty with code.

Set up our users

By default, jBPM is set up with a few test users who happen to be characters from Sesame Street: Ernie, Grover, and so on. Unless we want to give our brave testers a complex, we had better set them up with some proper usernames for the system. Of course, these usernames have to have a correspondence with the swimlanes that we used in the previous chapter to assign tasks. The relationship is one of "users" belonging to "groups" and it is modeled in the database that underpins jBPM. The behavior we are looking for is that when Jack Thompson, who is a Talent Scout, logs into the web console, he should see tasks in his task list relating to "Hold auditions", as he fulfils the swimlane criteria for that node in our process definition. We can set up new users by copying in our group and user information to the pre-existing database tables.

We don't need to set up every single user in the organization right now, only those people who will be helping us out with testing our proof-of-concept system. We'll need at least one user for each type of swimlane. We'll give our users suitable, system-ready usernames, without any spaces or non-standard characters: this will make the upgrade path to fully integrate with a Windows domain easier in the long run. One consideration we do have to bear in mind: at run time, we won't know who some of our users will be, as they are actually defined during process execution. For example, we don't know who our band members are going to be until we've held auditions and selected the band. We can get round this by simply creating anonymous logins that whoever gets picked can use.

For our purposes, we’ll set up the following groups and users:

- Talent scout — powellb
- Legal adviser — rumpoleh
- Band member — memberb
- Record producer — dredr
- Artist development — harrisr
- Songwriter — lennonj
- Musician — hendrixj
- Video production — welleso
- Artist — monetc
- Administrator — admin
- Manager — manager

In order to insert these users we need to get the database running. Start the JBoss application server with the shortcut we created in the last chapter. Once the application server has finished its start up routine, open up your web browser and go to http://localhost:8080/jmx-console/:

![JMX Agent View](image)

This is the management console that comes with the JBoss application server, which provides some utilities for managing it, including a simple database utility. We'll use this database utility to make the changes that we need. The database used in the default jBPM installation is called Hypersonic: it is a perfectly reasonable database for testing out jBPM on our local machine, although, in a later chapter, we'll see how we can swap it for a more robust database when we want to use our BPM system in anger.

In the JMX Console, under the jboss heading you will see a link entitled database=jbpmDB, service=Hypersonic: click this link. Scroll down the next page until you get to a line called void startDatabaseManager(). Click the Invoke button directly underneath. You will go to a new screen and after a few moments' delay the HSQL Database Manager will start:
On the left of the Database Manager, you can see the list of all the database tables used by jBPM. The tables we are interested in are PUBLIC.JBPM_ID_GROUP, PUBLIC.JBPM_ID_MEMBERSHIP, and PUBLIC.JBPM_ID_USER. If you want to see the existing contents of these tables, you can type this simple bit of SQL into the cursor window at the top and click **Execute SQL**:

```
SELECT * FROM PUBLIC.JBPM_ID_GROUP
```

This will bring back the list of groups that are already set up in jBPM. As you can see there are five columns in the table: ID, CLASS, NAME, TYPE, and PARENT. We won’t linger over the meaning of these columns; suffice it to say that the values in these columns define the relationships between the database tables that manage jBPM users. We need to add our own groups in, and again, we do this by running a bit of SQL on the database. You can either type the following in yourself or copy and paste in the code from the insert-groups.sql file that is in the code download for this chapter:

```sql
/*First we add in the security roles*/
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (301,'G','manager','security-role',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (302,'G','participant','security-role',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (303,'G','administrator','security-role',NULL)

/*Then we add in the organisations*/
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (201,'G','Talent scout','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (202,'G','Legal adviser','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (203,'G','Band member','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (204,'G','Record producer','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (205,'G','Artist development','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (206,'G','Songwriter','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (207,'G','Musician','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (208,'G','Video production','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (209,'G','Artist','organisation',NULL)
INSERT INTO PUBLIC.JBPM_ID_GROUP VALUES (210,'G','Administrator','organisation',NULL)
```

The prototype user interface

After you click the **Execute SQL** button, the Database Manager should come back with confirmation that the new groups have been added:

If you run the `SELECT * FROM PUBLIC.JBPM_ID_GROUP` command again, you will see that our new groups are now in the table:

---

Now we need to do another insert for our new users:

```sql
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (101,'U','powellb','powellb@bland.com','powellb')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (102,'U','rumpoleh','rumpoleh@bland.com','rumpoleh')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (103,'U','memberb','memberb@bland.com','memberb')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (104,'U','dredr','dredr@bland.com','dredr')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (105,'U','harrisor','harrisor@bland.com','harrisor')
```

The prototype user interface

INSERT INTO PUBLIC.JBPM_ID_USER VALUES (106,'U','lennonj','lennonj@bland.com','lennonj')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (107,'U','hendrixj','hendrixj@bland.com','hendrixj')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (108,'U','weloso','weloso@bland.com','weloso')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (109,'U','monetc','monetc@bland.com','monetc')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (110,'U','admin','admin@bland.com','admin')
INSERT INTO PUBLIC.JBPM_ID_USER VALUES (111,'U','manager','manager@bland.com','manager')

Again, you can either type in the command yourself or use the insert-users.sql file in the code download. Finally, we have to link our users to our groups through the PUBLIC.JBPM_ID_MEMBERSHIP table. We need to define all our users with a security role of "participant". In addition, our Administrator is given security roles of both "administrator" and "manager", and our Manager has a security role of "manager". We also need to assign each user to the swimlane groups that we have just added to the database. We do this by adding in new rows to the PUBLIC.JBPM_ID_MEMBERSHIP table, specifying the relationships between the rows in the PUBLIC.JBPM_ID_USER and PUBLIC.JBPM_ID_GROUP tables. We link the two tables' ID fields together: if you have deviated from the standard installation in any way, you will have to edit the following SQL to get the relationships right as specified above:

/*Then we make all our normal users participants*/
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'101','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'102','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'103','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'104','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'105','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'106','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'107','302')
INSERT INTO PUBLIC.JBPM_ID_MEMBERSHIP VALUES (NULL,'M',NULL,NULL,'108','302')

Phew, thankfully that's done, our proof-of-concept users are set up and ready to go. We can now safely close the Database Manager.
The prototype user interface

Deploy the process and user interface

Let's now deploy our process definition and console task forms to the server, so we can have a play around with them. First, start the application server again, if you closed it after the database changes we just made. Next, in the Designer, switch to the Deployment tab of the main editing window. You will be presented with a list of all the files that are available for deployment to the application server:

Make sure that all of the files in the left-hand pane are checked: by default, the files in the right-hand pane will be unchecked and we can leave it that way for now. In later chapters, we'll see how to deploy Java classes and resources through the right-hand pane.Scrolling down, we can see that we also have the opportunity here to specify the URL of the application server to which we should deploy our files:

It shouldn't be necessary to change the settings here, although, it is worthwhile clicking the Test Connection button to make sure the Designer can connect to the application server without any problems. If all is well, click the Deploy Process Archive button to deploy our files to the application server. This is all we need to do to put our process live.

One nice little feature of jBPM is that the engine is smart enough to keep track of the versions of our process that we deploy, without us having to manually provide version numbers or worry about overwriting previous versions of the process. For example, if I open up the Database Manager again and look in the PUBLIC.JBPM_PROCESSDEFINITION table at the versions of our "Produce music products" process that I have deployed to the application server I can see eight versions:

Every time I click the Deploy Process Archive button, a new version of the "Produce music products" process is added to the end of this table.
The prototype user interface

Investigating the web console interface

The jBPM web console is where our users will do their tasks, monitor the process, and administer the running of the live process. The web console is developed and maintained by the jBPM project team as an example of a web front-end to the jBPM process engine. The intention is that the example web console can serve as the starting point for jBPM users' own implementations of jBPM: it can be developed and tailored to our exact needs. Of course, if you do make improvements and develop the code base, you should submit your changes back to the jBPM community to help make the project even better.

At the time of writing, the web console is under very active development for the 3.2 release of jBPM, and hence the version that you actually end up using may well differ slightly from that presented in these pages. No matter, the concepts will remain the same and it will only be aesthetic differences, if there are any.

Let's have a look at the web console. With the application server running, fire up your browser and go to \[http://localhost:8080/jbpm-console/\] where you will be presented with the console login page:

In the middle of the screen, there is a hard-coded table featuring the usernames that are used with the default jBPM example process. We can ignore these and use one of our own logins that we put into the database, previously. Log in with the username and password of "powellb" and "powellb", to log in as our Talent scout. Once we're logged in, we can see the various portions of the web console that we can interact with, some of which are targeted at end users, some at managers, and some at administrators:

Of course, in a live environment we'd probably want to change this a bit so that end users don't get to see the managerial or administration sections, but we won't worry about that for now. Let's explore the various parts of the console. To make our exploration meaningful, let's create an instance of our process. Click Processes, then Start process next to the latest version of the "Produce music products" process:
The prototype user interface

This starts a new instance of the process and takes us to the first task form. If you now click Process Instances, you will see a list of all instances of the process that are currently in course on this server. If you click View you can see all the details of that process instance:

![Screen Shot of JBoss BPM Console Process Instance View]

From here, we can inspect the process definition diagram and code, as well as see some history of what has happened to this instance of the process while it has been running.

End users

End users will naturally be the heaviest users of the process's user interface, and there are two elements of the web console, in particular, that are expressly designed for them: tasks lists and task forms. The ordinary user should live and breathe in their task list as it represents the queue of work that they have to complete to fulfil their part of the process.

The web console actually contains two task lists, one for the user, but also one for the group. The user's task list presents a list of all the tasks that are currently assigned to the user that is logged in. For example, if we click User Task List in the menu we can see that we currently have one task assigned to us:

![User Task List](image)

This makes sense as we are currently logged in as a Talent scout, and we have just started an instance of the process where the first task node, "Hold auditions", is assigned to the Talent scout swimlane. If you were to log out and log in as a different user, you would see that their task list is currently empty, as they haven't had anything assigned to them yet.

You will also notice as you play around with the web console that tasks exist in either the user task list or the group task list, but not both. This is to make sure that you don't get multiple users working on the same group task: the idea is that the user "takes" a task off the group stack and adds it to their own to work on it. The group task list is intended for situations where you have a pool of users, any one of whom could take the task, removing it from the group task list and bringing it into their personal user task list.

The prototype user interface

If we go back to the user task list, and click View, next to the task we have assigned to us in our list, we are presented with the screens we need to be able to complete this task. This is the task form we generated previously, and we can now see how those XHTML elements are rendered in our browser:

![Task Instance View](image)

We can see that the web console has indeed picked up the "mapped names" of our process variables, using "Audition date" and "Audition location" as the labels in the input form. We can also see that the task form has three action buttons that the user can click: **Save**, **Cancel**, and **Save and Close**. The **Save** and **Cancel** buttons will be present on every task form page, although we will find that later in the process the **Save and Close** button will be called different things. This is because this button is the transitionButton element we saw earlier that will pick up the transition name, if one is specified. If there isn't a name specified for the transition, the button is simply labeled **End Task** by default. The **Save** button allows the user to fill out the task form and save its contents without submitting it and moving to the next node of the process. This is particularly useful in more complex forms where the user may not know exactly what they need to enter straight off. Similarly, the **Cancel** button allows the user to discard any input that they have made and go back to their task list without moving on in the process.

---

One of the best features of the web console, one which really highlights the benefits of business process management, is shown in the **Diagram** tab of the task form:

As you can see, this screen shows the user how their current task fits into the context of the process as a whole, by highlighting the current node in the process diagram. This is incredibly powerful from an end user's point of view as they can easily see how they are one cog in a bigger machine, and how the quality and timeliness of their work affects the work of others.

If we go back to the task form, enter some example data into the form and click **End Task**, the process variables we have entered are saved to the jBPM database, the task is marked as completed and the process execution moves on to the next node in the process definition. As the next task node is also assigned to our Talent scout, we can go back into the task list and pick up the next task:

The prototype user interface

Managers

The managerial part of the web console is probably the least developed part of the system at the moment. It is anticipated that managers will be able to use the web console to deploy new processes, monitor operation of the process, correct end users' process variable mistakes, cancel processes if need be, and monitor process timers. However, this functionality hasn't been fully developed at the time of writing; perhaps it will have been by the time you come to read this book. Nevertheless, never fear, as some of this functionality overlaps with the Business Activity Monitoring functionality that we will build in a later chapter.
If you log out of the web console and log back in with the username "manager" and password "manager", you can see that there is one more menu option available to our managers, "Deploy process":

This is actually an alternative way for us to deploy our processes, rather than using the deployment functionality in the Designer. Personally, I find it easier to use the Designer's deployer, but you might want to experiment with this one.

Of course, managers can also go into any of the task forms through the Process instances menu and use the re-assignment functionality to re-distribute the workload around their team:

Adapt the web console

The web console functions perfectly well as it is, although there are obviously a few things we'd like to do to make it our own. In later chapters, we'll make some further adaptations to the web console, but for now we'll content ourselves with adding some "help" text to the task form pages to give our proof-of-concept testers a few pointers as to how they should use the system. We are simply going to include a bit of text on each task form telling the users what is expected of them. We can do this with some very straightforward HTML. Go back into the Designer and double-click the *hold-auditions.xhtml* task form in the Package Explorer. Scroll down to the end of the code and add the following lines of HTML and text on a new line between the *dataform* and *uicomponent* closing tags:

```
<br /><br />
Add in the date and location of the next audition. Please enter the date in the format DD-MM-YYYY.
```

When you’ve finished click Save to save your changes for later, or click End Task to submit and move on. You can click Cancel if you don’t want to save your changes and are not ready to submit.

Hopefully, this code isn’t too earth shattering for you. If we now re-deploy the process definition to the server, start a new instance of this latest version of the process and have a look at the Hold auditions task form, we can see that our help text shows up on the screen to help our users understand what they need to do:

![Task Instance View](image)

You can do this for every task form where you believe it would be worthwhile to give the end user a bit of a helping hand. Remember, however, that if for some reason you have to re-generate the task form your help text will be lost.

**Sign off for the proof of concept**

All that remains now is to demonstrate the proof-of-concept web console to the sponsor and the proof-of-concept testers, and to ask them to sign it off as ready for testing. At this stage, it is very worthwhile setting some expectations and making it clear to these people that they are not signing off the system as being production-ready, merely that the bare functionality is in place to have a meaningful proof-of-concept test.

---

The prototype user interface

We also need to ask our proof-of-concept users to start gathering together the data they will need to put the system through its paces. They should look at historical records of the process to pull together this information: although, sometimes there will be no option but to fake it as the data may not have been recorded. As long as the data is sensible and realistic, there is no problem with this approach. The important thing is that the process definition and the user interface can be tested under semi-realistic conditions by the testers going through the process in "fast forward" with pre-determined data.

Summary

In this chapter, we have built the user interface that our proof-of-concept testers will use to interact with the process definition, which we built in the previous chapter. At this stage, we are not worrying about making the web console look pretty or in adapting it for our specific requirements: we are just going to make do with what is available out of the box. After all, the basic functionality of the jBPM web console is actually quite advanced and really provides more than enough for us to get started. In later chapters, we'll tinker here and there with it and we'll work at making the system more production-ready. But for now, we have our proof-of-concept system ready to go, and in the next chapter we'll see if we can indeed prove that concept.

We have covered:

- Building task forms
- Setting up users and groups
- Deploying the process
- The elements of the user interface
- Adding help text to the task forms
- Obtaining sign-off that the UI is ready to run the proof of concept

Our final deliverable from this phase has been the web console that has been signed off as fit for the purposes of running our proof of concept test.
Where to buy this book

Free shipping to the US, UK, Europe, Australia, New Zealand and India.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.