JPOS ™ Presentation Manager

Quick Guide
# Table Of Contents

1 Abstract................................................................................................................. 3
2 Copyright.................................................................................................................. 4
3 Quick guide............................................................................................................... 5
   3.1 Revision History................................................................................................. 5
   3.2 Before Starting................................................................................................. 6
   3.3 Starting.............................................................................................................. 7
      3.3.1 Location Files............................................................................................ 7
      3.3.2 Menu File.................................................................................................. 7
      3.3.3 Jetty File.................................................................................................... 8
      3.3.4 Application Resources............................................................................. 8
      3.3.5 QBean........................................................................................................ 8
      3.3.6 Template.................................................................................................... 9
      3.3.7 Startup Log.............................................................................................. 9
   3.4 Entities................................................................................................................ 11
      3.4.1 Weak-entity............................................................................................... 12
      3.4.2 Highlights................................................................................................. 12
   3.5 Operations......................................................................................................... 13
      3.5.1 Definition.................................................................................................... 13
      3.5.2 Configuration.............................................................................................. 13
      3.5.3 Context....................................................................................................... 13
      3.5.4 Predefined Operations............................................................................ 14
      3.5.5 Custom Operations................................................................................... 14
         3.5.5.1 Define the Action.................................................................................. 14
         3.5.5.2 Define the Action class........................................................................ 14
            3.5.5.2.1 org.jpos.ee.pm.struts.actions.ActionSupport................................. 15
            3.5.5.2.2 org.jpos.ee.pm.struts.actions.EntityActionSupport.................. 15
            3.5.5.2.3 org.jpos.ee.pm.struts.actions.RowActionSupport..................... 16
            3.5.5.2.4 org.jpos.ee.pm.struts.actions.SelectedAction............................ 16
         3.5.5.3 Define JSP page..................................................................................... 16
         3.5.5.4 Icon & Style........................................................................................... 16
         3.5.5.5 Naming and internationalization....................................................... 17
   3.6 Fields.................................................................................................................. 18
      3.6.1 Converters................................................................................................. 18
         3.6.1.1 Simple converters.................................................................................. 19
         3.6.1.2 Advanced converters............................................................................ 20
         3.6.1.3 Generic converters............................................................................... 20
         3.6.1.4 Custom converters............................................................................... 20
      3.6.2 Validators..................................................................................................... 22
         3.6.2.1 Predefined validators......................................................................... 22
         3.6.2.2 Custom Validators............................................................................... 22
   3.7 Security............................................................................................................... 23
   3.8 Monitors............................................................................................................ 26
      3.8.1 Introduction............................................................................................... 26
      3.8.2 Existing sources....................................................................................... 27
      3.8.3 Existing formatters................................................................................... 27
      3.8.4 Examples................................................................................................... 28
1 Abstract

This document is intended to be a fast guide to use jPOS Presentation Manager so, you will not have a
detailed description or justification of the methodology here. Before starting with this document please
read jPOS Project Guide and the jPOS Programmer's Guide. This document assumes that you have basic
knowledge about jPOS, jPOS-EE and the documents above mentioned. It is also recommended to have a
basic knowledge about JSP and Struts1.
2 Copyright

JPOS Copyright Notice

jPOS Project [http://jpos.org] Copyright (C) 2000-2010 Alejandro P. Revilla

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.
3 Quick guide

3.1 Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>13/06/10</td>
<td>J. Paoletti</td>
<td>1.0</td>
<td>Started document</td>
</tr>
<tr>
<td>02/07/10</td>
<td>J. Paoletti</td>
<td>1.1</td>
<td>Some corrections. Added external location.</td>
</tr>
<tr>
<td>04/07/10</td>
<td>J. Paoletti</td>
<td>1.2</td>
<td>Minor fixes. Added Operation Context description.</td>
</tr>
<tr>
<td>24/07/10</td>
<td>J. Paoletti</td>
<td>1.3</td>
<td>Adjustments on validators section. Added order property to entities.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added security section content. Added monitor section content.</td>
</tr>
<tr>
<td>29/07/10</td>
<td>J. Paoletti</td>
<td>1.4</td>
<td>Added chapter numeration</td>
</tr>
</tbody>
</table>
3.2 Before Starting

Get the project modules from here [http://github.com/jpaoletti/jPOS-Presentation-Manager](http://github.com/jpaoletti/jPOS-Presentation-Manager). You can do this by installing git and use the following commands:

```
git clone git://github.com/jpaoletti/jPOS-Presentation-Manager.git
```

in some folder and after that, make symbolic links for each jPOS-Presentation-Manager/modules subfolder to your project modules folder.

Also, the following jPOS modules are required:

- jetty6
- commons
- jpos
- constants

As soon as possible, jPOS-PM will be available as an opt module in google jPOS repositories.

Note that an example of the PM can be downloaded at [http://github.com/jpaoletti/jPOS-PM-Test](http://github.com/jpaoletti/jPOS-PM-Test) to show most of the functions.
### 3.3 Starting

As first step I highly recommend to create a new folder in your modules directory named “project_ui”. Inside it create the following folder structure:

```text
<table>
<thead>
<tr>
<th>modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ project_ui</td>
</tr>
<tr>
<td>→ src</td>
</tr>
<tr>
<td>→ deploy</td>
</tr>
<tr>
<td>→ cfg</td>
</tr>
<tr>
<td>→ entities</td>
</tr>
<tr>
<td>→ monitors</td>
</tr>
<tr>
<td>→ webapps</td>
</tr>
<tr>
<td>→ pm</td>
</tr>
<tr>
<td>→ templates</td>
</tr>
</tbody>
</table>
```

Once you have this folders, you must create this “must have” files.

#### 3.3.1 Location Files

Locations are a complex concept. It will be explained somewhere else. This file must have at least the following content.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE menu SYSTEM "locations.dtd">
<locations>
  <location id="pmstruts" class="org.jpos.ee.pm.struts.MenuItemLocationStruts"/>
  <location id="external" class="org.jpos.ee.pm.struts.MenuItemLocationExternal"/>
</locations>
```

#### 3.3.2 Menu File

This is the menu. Nesting is arbitrary and `perm` property only applies if you use `pm_security`.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE menu SYSTEM "menu.dtd">
<menu>
  <menu-list text="pm.menu.list.xxxx" perm="">
    <menu-list text="pm.menu.item.xxxx" perm="">
      <menu-item text="pm.menu.item.xxxx" perm="">
        <location id="pmstruts" value="/list.do?pmid=yyyy" />
      </menu-item>
      <menu-item text="pm.menu.item.yyyy" perm="">
        <location id="external" value="http://jpos.org" />
      </menu-item>
    </menu-list>
  </menu-list>
</menu>
```
Each “text” is an entry key in the application resource file. The “perm” is the permission needed to see this menu option when using security.

The location is the way the menu builds the menu. pm_struts has two locations, one for internal use of pm entities and one for external calls to another pages.

### 3.3.3 Jetty File

Jetty file defines port used by web application. Just copy

```plaintext
modules/jetty6/cfg/jetty.xml.sample
to
modules/project_ui/cfg/jetty.xml
```

### 3.3.4 Application Resources

This file is where all the text will live.

```plaintext
cfg/ApplicationResource.properties
```

```
main.title=Some Title
main.subtitle=Some Subtitle
...
```

### 3.3.5 QBean

The bean file must be created in the application deploy folder and is the “running” part of PM. The definition must be something like:

```xml
<presentation-manager class="org.jpos.ee.pm.struts.PMStrutsService" logger="Q2">
  <property name="debug" value="false" />
  <property name="template" value="default" />
  <property name="appversion" value="1.0.0" />
  <property name="login-required" value="true" />
  <property name="ignore-db" value="false" />
  <property name="title" value="main.title" />
  <property name="subtitle" value="main.subtitle" />
  <property name="default-data-access" value="org.jpos.ee.pm.core.DataAccessDB"/>
  <property name="persistence-manager" value="org.jpos.ee.pm.core.DBPersistenceManager"/>

  <!-- Business Entities -->
  <property name="entity" value="cfg/entities/xxxx.xml" />
  <!-- ... -->

  <!-- Monitors -->
  <property name="monitor" value="cfg/monitors/q2.xml" />
  <!-- ... -->
</presentation-manager>
```
<table>
<thead>
<tr>
<th>debug</th>
<th>Activate some debugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>template</td>
<td>Name of the folder containing the template to uses. Default uses modules/pm_struts/webapps/templates/default. A good practice is to copy this folder (full path) inside your project module to create your own template.</td>
</tr>
<tr>
<td>appversion</td>
<td>Version shown in main page of your application</td>
</tr>
<tr>
<td>login-required</td>
<td>If true the application uses authentication.</td>
</tr>
<tr>
<td>ignore-db</td>
<td>If true, the application ignores everything related to Data Base access.</td>
</tr>
<tr>
<td>title</td>
<td>ApplicationResource key for the title</td>
</tr>
<tr>
<td>subtitle</td>
<td>ApplicationResource key for the subtitle</td>
</tr>
<tr>
<td>default-data-access</td>
<td>Default entity Data Access</td>
</tr>
<tr>
<td>entity*</td>
<td>One or more entities files</td>
</tr>
<tr>
<td>monitor*</td>
<td>One or more monitor files</td>
</tr>
</tbody>
</table>

### 3.3.6 Template

It is recommended to copy the default template (located at pm_struts module modules/pm_struts/webapps/pm/templates/default) into your own module, preserving the path but changing the name of the final folder. Something like:

```
cp modules/pm_struts/webapps/pm/templates/default
modules/project_ui/webapps/pm/templates/mytemplate
```

You should configure your Qbean file as

```xml
<property name="template" value="mytemplate" />
```

After that, you can freely change the style of the site by modifying the css and images inside your folder.

### 3.3.7 Startup Log

When starting, Q2 server will show up some log for presentation manager. This log is something like:

```
<log realm="org.jpos.ee.pm.struts.PMStrutsService" ...>
<info>
  <startup>Presentation Manager activated</startup>
  <configuration>
    (*) Template    mytemplate
    (*) Default Data Access org.jpos.ee.pm.core.DataAccessDB
    (*) Application version 1.0.0.0
    (*) Title        pm.title
    (*) Subtitle     pm.subtitle
  </configuration>
```

Jeronimo Paoletti - jPOS Presentation Manager Quick Guide
(*) Contact mailto:jpaoletti@angras.com.ar
(*) Login Required false
(*) Persistance Manager org.jpos.ee.pm.core.XXX
<configuration>
<entities>
  (*)(*) sysconfig org.jpos.ee.SysConfig
  ...
</entities>
<monitors>
  (*)(*) q2 org.jpos.ee.pm.core.monitor.FileMonitorSource
  (*)(*) status org.jpos.ee.pm.core.monitor.SQLMonitorSource
  ...
</monitors>
<locations>
  (*)(*) pmstruts org.jpos.ee.pm.struts.MenuItemLocationStruts
  (*)(*) external org.jpos.ee.pm.struts.MenuItemLocationExternal
</locations>
</info>
</log>

Reference:

(*) Item correctly loaded.
(·) Weak entity correctly loaded.
(?) Item correctly loaded but with an unknown class.
(!) Item with error. Look for the exception in previous log.
### 3.4 Entities

The entities files are recommended to go in `cfg/entities` of your ui module. General definition of an entity is:

```xml
<?xml version='1.0' ?>
<!DOCTYPE entity SYSTEM "cfg/entity.dtd">
<entity id="anentity" clazz="org.jpos.ee.SomeClass" extendz="otherentity" no-count="false">
  <dataAccess class="org.jpos.ee.pm.editor.DataAccessEntity" />
  <listFilter class="xxx" />
  <order>...</order>
  <highlights>...</highlights>
  <operations>...</operations>
  <field.../>
</entity>
```

<table>
<thead>
<tr>
<th>id</th>
<th>Id of the entity. Must be global unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>clazz</td>
<td>Class of the represented object</td>
</tr>
<tr>
<td>extendz</td>
<td>Optional entity id to extends its fields</td>
</tr>
<tr>
<td>no-count</td>
<td>If true, no operation use the “count” method of the data access. It has been made for entities with a large amount of instances.</td>
</tr>
<tr>
<td>dataAccess</td>
<td>Overrides the default data access.</td>
</tr>
<tr>
<td>operations</td>
<td>A list of available operations for this entity.</td>
</tr>
<tr>
<td>field*</td>
<td>The list of fields of the entity.</td>
</tr>
<tr>
<td>ListFilter</td>
<td>Is an implementation of EntityFilter interface to filter the content of the list operation. In a near future, this will be a property of the operation list.</td>
</tr>
<tr>
<td>Order</td>
<td>String with a list of field ids that determine the order of the fields. May be separated with any separator. If a field is not included it goes to the tail.</td>
</tr>
</tbody>
</table>

The name of the entity is determined by the following line in the application resource file:

```
pm.entity.entityid=Some name
```

It is recommended to save the file with the id of the entity as the filename. In this case

```
cfg/entities/entityid.xml
```

Before an entity become available, you must add the following line in deploy/80_pm.xml file

```
<property name="entity" value="cfg/entities/entityid.xml" />
```
3.4.1 Weak-entity

A weak-entity is an entity hardly dependent of another entity. That means that it only exists if the other entity contains it and cannot exists alone.

```xml
<entity id="..." clazz="...">
    <owner>
        <entityId>xxxxx</entityId>
        <entityProperty>xxxxx</entityProperty>
        <entityCollectionClass>java.util.ArrayList</entityCollectionClass>
        <localProperty>xxxxx</localProperty>
        <localPosition>xxxxx</localPosition>
    </owner>
    ....
</entity>
```

<table>
<thead>
<tr>
<th>entityId</th>
<th>Parent entity id</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityProperty</td>
<td>Parent entity field that points to a collection of weak entity instances</td>
</tr>
<tr>
<td>entityCollectionClass</td>
<td>Class of the entity parent collection of weak entity instances</td>
</tr>
<tr>
<td>localProperty</td>
<td>Optional, entity field pointing to parent</td>
</tr>
<tr>
<td>localPosition</td>
<td>Optional, entity field index for parent collection. Only needed in indexed collections.</td>
</tr>
</tbody>
</table>

3.4.2 Highlights

Highlights are a way to make some items stress on others on any operation but specially in list.

```xml
<entity id="..." clazz="...">
    <highlights>
        <highlight field="xxx" value="xxx" color="xxx" scope="xxx" />
    </highlights>
</entity>
```

<table>
<thead>
<tr>
<th>field</th>
<th>Id of the field that will be checked for highlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value to determine if the highlight is applied</td>
</tr>
<tr>
<td>color</td>
<td>Color (in HTML format)</td>
</tr>
<tr>
<td>scope</td>
<td>Could be “instance” (highlight all the instance) or “property” (just highlight the field)</td>
</tr>
</tbody>
</table>
3.5 Operations

3.5.1 Definition

Operations are the “things” you can do with an entity instance. For example, you can list the instances, add a new one, edit an existing, delete one, sort the list, filter the list and so on.

Each operation may have its own properties and may show a confirmation dialog before execute.

```xml
<entity id="..." clazz="...">
<operations>
    <operation id="xxx" scope="xx" display="xx" enabled="xx" confirm="xx">
        <context class="xxx"/>
        <properties>
            <property name="xxx" value="xxx"/>
            ...
        </properties>
    </operation>
    ...
</operations>
...
</entity>
```

3.5.2 Configuration

<table>
<thead>
<tr>
<th>id</th>
<th>Unique global id. The same operation may be used on more than one entity but just once on the same entity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope</td>
<td>item</td>
</tr>
<tr>
<td>display</td>
<td>Operations id where this operation will be shown</td>
</tr>
<tr>
<td>enabled</td>
<td>true</td>
</tr>
<tr>
<td>confirm</td>
<td>true</td>
</tr>
<tr>
<td>properties</td>
<td>Operation specific properties</td>
</tr>
</tbody>
</table>

3.5.3 Context

A context may be defined for each operation. A context is a class implementing OperationContext or extending OperationContextSupport with three methods.

```java
/** This method is executed at the very beginning of the process, before converting or replace any data on objects. */
public void preConversion (PMContext ctx) throws PMException;

/** This method is executed before trying to execute the main method of the operation, that is before opening any transaction. */
* @param ctx The context
```
public void preExecute (PMContext ctx) throws PMException;

/** This method is executed after the main method of the operation.
 * @param ctx The context
 */
public void postExecute (PMContext ctx) throws PMException;

3.5.4 Predefined Operations

- **list.** Have the following properties
  - **searchable:** Add a little search
  - **pagination:** Add pagination for the list
  - **show-row-number:** Shows a little number near each row
  - **operation-column-width:** Width of the first column of the list table
  - **sort-field:** Id of the field taken for sorting
  - **sort-direction:** asc / desc
- show
- add
- edit
- delete
- sort
- filter

3.5.5 Custom Operations

In *pm_core*, operations are just a definition. The PM implementation must define a way to “execute” this operations. In *pm_struts*, each operation is defined as an action with the path defined as the operation id. To define a new operation (let’s say, named myoper) in *pm_struts* you need to do the following steps.

3.5.5.1 Define the Action

You need to create the following file in your module and append the following text.

```xml
<action path="/myoper" type="org.jpos.ee.pm.struts.actions.MyOperAction">
  <forward name="success" path="/pages/myoper.jsp"/>
</action>
```

This will add the new action. If you need to define an FormBean (remember that this is Struts 1) you can add it in the file *cfg/struts.bean*. If you want to add a global forward, you can do it in *cfg/struts.fwds*.

3.5.5.2 Define the Action class

This could be not necessary as your action can be defined as a forward, but, if you need to, there are some classes to inherit from. In any case, you need to implement the following method:
protected abstract void doExecute(PMStrutsContext ctx) throws PMException;

If this method ends without errors, then “success” forward is returned. If you want to return another forward, you can throw the following exception:

throw new PMForwardException("someforward");

The difference between super classes are the helper methods and the checks that can be done automatically.
Also you will always get a PMStrutsContext with (at least) the following content:

ctx.getRequest() // return the ServletRequest
ctx.getResponse() // return the ServletResponse
ctx.getErrors() // return an error array list to put the errors (PMMMessage)
ctx.get(DB) // return a DB object if database is used. Requires eecore3min.
ctx.getUser() // return the logged user.

3.5.5.2.1 org.jpos.ee.pm.struts.actions.ActionSupport

Using this class you can redefine the following method to ignore the existence of a logged user.

//Default is true
protected boolean checkUser(){
    return false;
}

3.5.5.2.2 org.jpos.ee.pm.struts.actions.EntityActionSupport

This class mainly add entity related information to the context and embed the doExecute inside a data access transaction.

/** Opens a Data Access transaction before doExecute */
protected boolean openTransaction() { return false; }
/** Makes the operation generate an auditory entry. Not implemented yet */
protected boolean isAudited() { return true; }
/** Forces execute to check if there is an entity defined in parameters */
protected boolean checkEntity(){ return true; }

This class also adds the following items into the context:

cx.getEntityContainer() //return an EntityContainer that is a container for each entity and his session related content, like the actual list, the selected items and the actual filter.
cx.getEntity() //return the entity inside the container
cx.getOperation() //return a reference to the actual operation.
3.5.5.2.3  org.jpos.ee.pm.struts.actions.RowActionSupport

This class adds the selection of an specific instance of the entity.

```java
/** Throws an exception if selected item does not exists */
public boolean testSelectedExist(){ return true; }
```

This class change the `ctx.getSelected()` item. It also add the following methods to process the instance fields:

```java
protected void processField(PMStrutsContext ctx, Field f, EntityInstanceWrapper wrapper) throws PMException;
private boolean validateField(PMStrutsContext ctx, Field field, EntityInstanceWrapper wrapper, Object o) throws PMException;
```

3.5.5.2.4  org.jpos.ee.pm.struts.actions.SelectedAction

This class is not intended to extend but to use directly into the action file as the Action. It do nothing and it is the way to say “jsp page does all the work”.

3.5.5.3  Define JSP page

It is not always necessary, some action just “do something” without showing nothing but, in the case you need to show a result, you will need to define a JSP page. This must be defined in the following folder:

`modules/project_ui/webapps/pm/pages/myoper.jsp`

You can take any of the pm_struts pages as an example. You can also define your own custom tags in:

`modules/project_ui/webapps/pm/WEB-INF/tags`

You have also all the struts and JSTL tags by using:

```html
<%@ taglib uri="/WEB-INF/tld/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/tld/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/tld/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/tld/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/tld/fn.tld" prefix="fn" %>
<%@ taglib uri="/WEB-INF/tld/fmt.tld" prefix="fmt" %>
<%@ taglib tagdir="/WEB-INF/tags" prefix="pm" %>
...
```

Expression Language (EL) is also available, so you can use things like `${entity} ${field} ${operation}` in your JSP.

3.5.5.4  Icon & Style

To add an icon associated to your action you must add a 16x16 transparent gif into the following folder:

`modules/project_ui/webapps/pm/templates/mytemplate/img/myoper.gif`
After that, you need to add in the buttons.css the following lines:

```css
.modules/project_ui/webapps/pm/templates/mytemplate/buttons.css

.myoper {
  background: url(img/myoper.gif) no-repeat 10px 8px;
}
```

3.5.5.5 Naming and internationalization

Finally, you need to add a name to your operation. You can do this by adding the following line in your application resource file:

```javascript
operation.myoper=Oper name
```
3.6 Fields

Fields are the representation of each instance variable of the represented class that you are going to visualize in each operation.

```xml
<entity id="..." clazz="...">
  <field id="xx" property="xx" display="xx" align="xx" width="xx">
    <converters>
      ...
    </converters>
    <validators>
      ...
    </validators>
  </field>
  ...
</entity>
```

<table>
<thead>
<tr>
<th>id</th>
<th>Unique id for this field in this entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>Optional, the dot represented name of the property that must have a getter and setter in the represented class. This could be something like “description” or “address.number”. If not defined, id is taken for this purpose.</td>
</tr>
<tr>
<td>display</td>
<td>Optional, a white separated list of operation ids where this field will be shown. “all” is the default.</td>
</tr>
<tr>
<td>align</td>
<td>Optional, left (default)</td>
</tr>
<tr>
<td>width</td>
<td>Optional, mainly used by list operation, this is the width of the column for this field.</td>
</tr>
</tbody>
</table>

The name of the field is determined by the following line in the application resource file:

```
pm.field.entityid.fieldid=Some name
```

3.6.1 Converters

The converters for a field are the way it is visually shown in the site and also the way the visual representation is translated to a Java object. Each field may have more than one converter for several operations. General definition is:

```xml
<field id="xx" ...>
  <converters>
    <converter class="xxx" operations="xxx">
      <properties>
        <property name="xxx" value="xxx" />
        ...
      </properties>
    </converter>
  </converters>
</field>
```
<table>
<thead>
<tr>
<th><strong>class</strong></th>
<th>Class of the converter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>operations</strong></td>
<td>A white separated list of operation ids where this converter will be used for this field.</td>
</tr>
<tr>
<td><strong>properties</strong></td>
<td>A group of specific properties for this converter.</td>
</tr>
</tbody>
</table>

All the converters have this default properties:

<table>
<thead>
<tr>
<th><strong>pad-count</strong></th>
<th>If padding is applied, this is the number of characters to fill</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pad-char</strong></td>
<td>If padding is applied, this is the character used to fill the text</td>
</tr>
<tr>
<td><strong>pad-direction</strong></td>
<td>If padding is applied, this is the direction of the pad. Could be left or right</td>
</tr>
<tr>
<td><strong>prefix</strong></td>
<td>A text shown before the result text</td>
</tr>
<tr>
<td><strong>suffix</strong></td>
<td>A text shown after the result text</td>
</tr>
</tbody>
</table>

### 3.6.1.1 Simple converters

Some of the most common converters are the following

```xml
<converter class="org.jpos.ee.pm.struts.converter.EditStringConverter">
  <properties>
    <property name="max-length" value="xx" />
  </properties>
</converter>

<converter class="org.jpos.ee.pm.struts.converter.ShowBooleanConverter" />

<converter class="org.jpos.ee.pm.struts.converter.EditBooleanConverter">
  <properties>
    <property name="with-null" value="true | false (default)" />
  </properties>
</converter>

<converter class="org.jpos.ee.pm.struts.converter.EditLongConverter" />

<converter class="org.jpos.ee.pm.struts.converter.EditIntegerConverter" />

<converter class="org.jpos.ee.pm.converter.ShowDateConverter">
  <properties>
    <property name="format" value="dd/MM/yyyy HH:mm:ss" />
  </properties>
</converter>

<converter class="org....converter.ShowISODumpConverter" />

<converter class="org....converter.ShowLocalizedStringConverter"/>

<converter class="org....converter.EditPasswordConverter"/>
```
3.6.1.2 Advanced converters

There are more advanced converters to manipulate collections, objects and weak entities.

```xml
<converter class="org.jpos.ee.pm.struts.converter.EditCompositionConverter">
<!-- This is for editing weak entities. Just take a look :) -->
    <properties>
        <property name="weak-entity" value="xx" />
    </properties>
</converter>

<converter class="org.jpos.ee.pm.struts.converter.ShowCompositionConverter">
<!-- display a table with the instances of the other entity, those fields displayed on 'list' operation -->
    <properties>
        <property name="weak-entity" value="xx" />
    </properties>
</converter>

<converter class="org....converter.EditSingleAggregationConverter">
<!-- display a combo with the instances of the other entity-->
    <properties>
        <property name="entity" value="xxx" />
        <property name="with-null" value="xxx" />
    </properties>
</converter>
```

3.6.1.3 Generic converters

Generic converter is a converter defined by a bash based xml usually located at cfg/converters/. It is used as every converter but the property “filename” must be defined.

```xml
<converter class="org.jpos.ee.pm.converter.GenericConverter">
    <properties>
        <property name="filename" value="cfg/converters/show.tostring.converter" />
    </properties>
</converter>

<converter class="org.jpos.ee.pm.converter.GenericConverter">
    <properties>
        <property name="filename" value="cfg/converters/show.decimal.converter" />
        <property name="prefix" value="$ " />
        <property name="suffix" value=" .-" />
        <property name="format" value="0" />
    </properties>
</converter>
```

3.6.1.4 Custom converters

There are 2 ways to define a new converter. One is through a class extending Converter and define this 2 methods:

```java
public String visualize(PMContext ctx) throws ConverterException;
public Object build(PMContext ctx) throws ConverterException;
```

Visualize in pm_struts usually returns a JSP page name, which must be created in the following folder.
Build must take a “string” representation of the object and turn it into the final object. For example, Integer Converter turns a “5” into the integer representation, 5.

modules/project_ui/webapps/pm/converters

Usually, “ShowSomethingConverter” does not need to build nothing so you can just throw an IgnoreConversionException().

Both methods has the following entries in context and helpers inherited by Converter:

<table>
<thead>
<tr>
<th>Field</th>
<th>ctx.get(PM_FIELD);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Value</td>
<td>ctx.get(PM_FIELD_VALUE);</td>
</tr>
<tr>
<td>Instance Wrapper</td>
<td>ctx.getSelected();</td>
</tr>
<tr>
<td>Entity</td>
<td>ctx.getEntity();</td>
</tr>
<tr>
<td>Operation</td>
<td>ctx.getOperation();</td>
</tr>
<tr>
<td>Getter for properties</td>
<td>public String getConfig (String name, String def) ;</td>
</tr>
<tr>
<td>Getter for value</td>
<td>protected Object getValue(Object instance, Field field);</td>
</tr>
<tr>
<td>Add generic properties</td>
<td>public String visualize(Object obj, String extra);</td>
</tr>
</tbody>
</table>

The other way is to create a generic converter file, which has the following definition:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<generic-converter>
    <visualize>
        return value.toString();
    </visualize>
    <build>
        throw new IgnoreConversionException("");  
    </build>
</generic-converter>
```

Both visualize and build are the same than a normal converter taking in mind that:

- Content is a [Bean Shell](https://bean-shell.sourceforge.net/) script
- You have a “value” variable with the value of the field that you are converting. Its an Object.
- You have a “converter” variable to access all the properties and helper methods of the Converter class.
- Both definition must return the same value as a normal converter method.

This way is pretty useful to introduce new converters in a production server without compile or restarting the server.
3.6.2 Validators

Validators are classes that check for unwanted situations, like invalid characters, out of range values and so on. A validator can be defined at field level or at operation level. Actually, a validator at field level will apply in all operations.

```xml
<field id="xx" ...>
  <!-- <operation id="xx"> -->
  <validator class="org.jpos.ee.pm.validator.XxValidator">
    <properties>
      <property name="xxx" value="xxx" />
    </properties>
  </validator>
  <!-- </operation> -->
</field>
```

A field may have more than one validator and each validator may have its own properties. Usually, each validator has a “msg” property that is a key of the resource file to show in an error box when the abnormal situation occurs.

### 3.6.2.1 Predefined validators

All of the following classes are in `org.jpos.ee.pm.validator` package

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsNameValidator</td>
<td>Validate the pattern “name”</td>
</tr>
<tr>
<td>LengthValidator</td>
<td>Uses the properties “max-length”, “max-length-msg”, “min-length” and “min-length-msg” to validate a string length.</td>
</tr>
<tr>
<td>NotNull</td>
<td>Check if the object is not null</td>
</tr>
</tbody>
</table>

### 3.6.2.2 Custom Validators

To create a custom validator you just need to implement `org.jpos.ee.pm.validator.Validator` interface but a better way is to inherit from `org.jpos.ee.pm.validator.ValidatorSupport` which allow the use of properties.

The validation method that must be implemented is

```java
public ValidationResult validate(PMContext ctx);
```

Context content is the same in converters. `ValidationResult` is a class with 2 variables

<table>
<thead>
<tr>
<th>boolean successful</th>
<th>True when validation was ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;PMMessage&gt; messages</td>
<td>List of errors to be shown</td>
</tr>
</tbody>
</table>

A `PMMessage` has the following content:

<table>
<thead>
<tr>
<th>key</th>
<th>Property for ActionErrors.add() method</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message from application resource file</td>
</tr>
<tr>
<td>Arg0, arg1, arg2, arg3</td>
<td>Arguments for the message</td>
</tr>
</tbody>
</table>

Each `PMMessage` will be placed as a struts error message.
3.7 Security

Security is, at the moment, implemented in a simple way based on eecore3 user management. Basically, it has the same “core + implementation” concept than Presentation Manager. Implementing an interface you can define your own security or use the basic, database based security or not use at all.

To use jPOS PM security you need to include pm_security_core module and one implementation. The DB based implementation is at pm_security_db and also there is a PM based module that provides a basic but useful set of entities to directly include in your PM bean (80_pm.xml) at pm_security_ui.

To make use of this modules, you must create the following file in your module_ui/deploy folder:

```
79_security.xml
<security-manager class="org.jpos.ee.pm.security.core.PMSecurityService"
logger="Q2">
  <property name="connector" value="org.jpos.ee.pm.security.db.PMSecurityDBConnector" />
</security-manager>
```

The connector define the main interface to implement a security instance module. After including security bean, you must activate login in presentation manager bean:

```
80_pm.xml
<property name="login-required" value="true" />
<property name="ignore-db" value="false" />
```

If you want to use the pm_security_ui entities you may also include the following lines to PM bean.

```
<!-- Security Entities -->
<property name="entity" value="cfg/entities/secuser.xml" />
<property name="entity" value="cfg/entities/secusergroup.xml" />
<property name="entity" value="cfg/entities/secperm.xml" />
<property name="entity" value="cfg/entities/secuserprofile.xml" />
```

Finally, you can use the following SQL script in your database. This will create a default “super” user with “test” password.

```
SET FOREIGN_KEY_CHECKS=0;
CREATE TABLE IF NOT EXISTS `sec_groups` (
  `id` bigint(20) NOT NULL auto_increment ,
  `name` varchar(32) NOT NULL,
  `description` varchar(255) default NULL ,
  `active` char(1) NOT NULL default 'Y' ,
  `creation` timestamp NOT NULL default CURRENT_TIMESTAMP ,
  PRIMARY KEY (`id`) )
```

Jeronimo Paoletti - jPOS Presentation Manager Quick Guide 23 / 29
```sql
CREATE TABLE IF NOT EXISTS `sec_groups` (`id` bigint(20) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `description` varchar(255) default NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

INSERT INTO `sec_groups` (`id`, `name`, `description`) VALUES (1, 'Super Administrators', 'Super Administrators', 'Y', CURRENT_TIMESTAMP);

CREATE TABLE IF NOT EXISTS `sec_perms` (
  `id` bigint(20) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `description` varchar(255) default NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

INSERT INTO `sec_perms` (`id`, `name`, `description`) VALUES (1, 'useradmin', 'User Administrator'),(2, 'login', 'Login');

CREATE TABLE IF NOT EXISTS `sec_group_perms` (
  `sec_group` bigint(20) NOT NULL,
  `sec_perm` bigint(20) NOT NULL,
  PRIMARY KEY (`sec_group`,`sec_perm`),
  KEY `sec_perm` (`sec_perm`)
) ENGINE=InnoDB;

INSERT INTO `sec_group_perms` (`sec_group`, `sec_perm`) VALUES (1, 1), (1,2);

CREATE TABLE IF NOT EXISTS `sec_users` (
  `id` bigint(20) NOT NULL auto_increment,
  `nick` varchar(32) NOT NULL,
  `password` varchar(32) default NULL,
  `name` varchar(128) default NULL,
  `active` char(1) default NULL,
  `deleted` char(1) default NULL,
  `email` varchar(255) default NULL,
  `change_password` char(1) NOT NULL default 'N',
  PRIMARY KEY (`id`),
  UNIQUE KEY `nick` (`nick`)
) ENGINE=InnoDB;

INSERT INTO `sec_users` (`id`, `nick`, `password`, `name`, `active`, `deleted`, `email`, `change_password`) VALUES (1, 'super', '6a817ee3201dab473f2406d3d9f7c061', 'administrador', 'Y', 'N', '', 'N');

CREATE TABLE IF NOT EXISTS `sec_user_groups` (
  `sec_user` bigint(20) NOT NULL,
  `sec_group` bigint(20) NOT NULL,
  PRIMARY KEY (`sec_user`,`sec_group`),
  KEY `sec_group` (`sec_group`)
) ENGINE=InnoDB;

INSERT INTO `sec_user_groups` (`sec_user`, `sec_group`) VALUES (1, 1);

CREATE TABLE IF NOT EXISTS `sec_user_props` (
  `id` bigint(20) NOT NULL default '0',
  `propValue` varchar(255) default NULL,
  `propName` varchar(32) NOT NULL default '',
  PRIMARY KEY (`id`),
  KEY `FK3A78989CA043F41A` (`id`)
) ENGINE=InnoDB;

ALTER TABLE `sec_group_perms`
  ADD CONSTRAINT `sec_group_perms_ibfk_1` FOREIGN KEY (`sec_group`) REFERENCES
```
Result will be a login screen at startup and “profile” link when logged as a user. With security activated, now you can make use of the “perm” attribute of a menu item to show / hide items depending on defined permissions of each user.
3.8 Monitors

3.8.1 Introduction

Monitors are still under extension but main idea is to “watch something over time”, like a DB table or a file. Basic definition, similar to entities is an xml with a line attached to PM bean:

```xml
80_pm.xml

...<property name="monitor" value="cfg/monitors/q2.xml" />
...```

A monitor definition is something like

```xml
<?xml version='1.0' ?>
<!DOCTYPE entity SYSTEM "cfg/monitor.dtd">
<monitor>
  <id>monitorId</id>
  <source class="org.jpos.ee.pm.core.monitor.XxMonitorSource">
    <properties>
      <property name="xx" value="yy" />
    </properties>
  </source>
  <formatter class="org.jpos.ee.pm.core.monitor.XxFormatter">
    <properties>
      <property name="xx" value="yy" />
    </properties>
  </formatter>
  <delay>xx</delay>
  <max>xx</max>
  <cleanup>true | false</cleanup>
  <all>true | false</all>
</monitor>
```

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong></td>
<td>Id of the monitor (used on menu or anywhere it need to be referenced)</td>
</tr>
<tr>
<td><strong>source</strong></td>
<td>Source is class extending abstract class MonitorSource and represents the “where monitor takes information”</td>
</tr>
<tr>
<td><strong>formatter</strong></td>
<td>Formatter is a class extending MonitorFormatter (or even MonitorFormatter class) that get a MonitorLine generated by source and turn it into a string.</td>
</tr>
<tr>
<td><strong>delay</strong></td>
<td>Delay between each refresh (in ms)</td>
</tr>
<tr>
<td><strong>max</strong></td>
<td>Maximum number of visible lines.</td>
</tr>
<tr>
<td><strong>Cleanup</strong></td>
<td>False by default, If true, each refresh of the monitor clean ups the list.</td>
</tr>
<tr>
<td><strong>All</strong></td>
<td>False by default, If true, each refresh of the monitor shows all the information given by the source</td>
</tr>
</tbody>
</table>
3.8.2 Existing sources
There are two predefined monitor source implementation.

<table>
<thead>
<tr>
<th>Module</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_core</td>
<td>org.jpos.ee.pm.core.monitor.FileMonitorSource</td>
<td>A monitor to watch a file. “filename” property is required.</td>
</tr>
<tr>
<td>pm_core_db</td>
<td>org.jpos.ee.pm.core.monitor.SQLMonitorSource</td>
<td>A monitor to watch a database query. Three properties are needed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• query: SQL text to get the monitor information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• last-line-query: SQL text to get the monitor last line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• id-column: column index which identifies the above query.</td>
</tr>
</tbody>
</table>

3.8.3 Existing formatters
There are two predefined monitor formatters.

<table>
<thead>
<tr>
<th>Module</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_core</td>
<td>org.jpos.ee.pm.core.monitor.MonitorFormatter</td>
<td>Simple formatter that receive a plain string and returns it “as is”</td>
</tr>
<tr>
<td>pm_core</td>
<td>org.jpos.ee.pm.core.monitor.BasicObjectArrayFormatter</td>
<td>Formatter that receive an array of items and uses two properties to format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• pads: A string separated by “#” that represents the padding of each array item. See the example on next section for details.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• separator: a string that separates each array item.</td>
</tr>
</tbody>
</table>
3.8.4 Examples

Monitor for q2.log file

```xml
<?xml version='1.0' ?>
<!DOCTYPE entity SYSTEM "cfg/monitor.dtd">
<monitor>
    <id>q2</id>
    <source class="org.jpos.ee.pm.core.monitor.FileMonitorSource" >
        <properties>
            <property name="filename" value="log/q2.log" />
        </properties>
    </source>
    <formatter class="org.jpos.ee.pm.core.monitor.MonitorFormatter" />
    <delay>3000</delay>
    <max>100</max>
</monitor>
```

Q2 LOG MONITOR (MONITOR)

```
<threads>
  <!-- name-registrar -->
  logger Q2: org.jpos.util.Logger
  presentation-manager: org.jpos.ee.pm.struts.PMStrutsService
  logger.pm-logger: org.jpos.util.Logger
</threads>
```

Jeronimo Paoletti - jPOS Presentation Manager Quick Guide
Monitor file for “status” table of jPOS **status** module.

```xml
<?xml version='1.0' ?>
<!DOCTYPE entity SYSTEM "cfg/monitor.dtd">  
<monitor>
  <id>status</id>
  <source class="org.jpos.ee.pm.core.monitor.SQLMonitorSource" >
    <properties>
      <property name="query" value="SELECT id, name, state, detail, lastTick, timeout, timeoutState FROM status" />
      <property name="last-line-query" value="SELECT id, name, state, detail, lastTick, timeout, timeoutState FROM status" />
    </properties>
  </source>
  <formatter class="org.jpos.ee.pm.core.monitor.BasicObjectArrayFormatter">
    <properties>
      <property name="pads" value="[0,R,,20] # [1,R,,40] # [2,R,,2] # [3,R,,75]" />
      <property name="separator" value=" | " />
    </properties>
  </formatter>
  <delay>3000</delay>
  <max>100</max>
  <cleanup>true</cleanup>
  <all>true</all>
</monitor>
```

**pads formats contains 4 parts, “[0, R, , 20]”**

1. Index of the array  
2. R or L: right or left padding  
3. Padding character  
4. Padding length