

PORTIONS (PORTlet actIONS)

User Guide

Controller

Specification of the application's controller

portlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd"
    version="1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet
-app_1_0.xsd">
    <portlet>
        <description>
            Sample portlet description
        </description>
        <portlet-name>
            SamplePortlet
        </portlet-name>
        <display-name>
            Sample Portlet
        </display-name>
        <portlet-class>
            net.sf.portions.controller.PortletController
        </portlet-class>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
            <portlet-mode>EDIT</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <supported-locale>en</supported-locale>
        <portlet-info>
            <title>Sample Portlet</title>
            <short-title>SamplePortlet</short-title>
            <keywords>List of keywords</keywords>
        </portlet-info>
    </portlet>
</portlet-app>
```

Configuration of the controller

portlet-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-config>
    <form-beans>
        <form-bean name="sampleForm"
            type="net.sf.portions.sample.form.SamplePortletForm"
        />
    ...
</portlet-config>
```

```

</form-beans>
<global-forwards>
  <forward name="success"
    mode="view"
    path="/jsp/view_1.jsp"
    content="text/html"/>
  <forward name="failure"
    mode="view"
    path="/jsp/view_1.jsp"
    content="text/html"/>
  ...
</global-forwards>
<action-mappings>
  <action name="step1"
    type="net.sf.portions.sample.action.Step1PortletAction
">
    <forward name="success"
      mode="view"
      path="/jsp/view_1.jsp"
      content="text/html"/>
  </action>
  <action name="step2"
    type="net.sf.portions.sample.action.Step2PortletAction
"
    input="/jsp/view_1.jsp"
    content="text/html"
    formName="sampleForm">
    <forward name="success"
      mode="view"
      path="/jsp/view_2.jsp"
      content="text/html"/>
  </action>
  ...
</action-mappings>
<message-resources parameter="resources.ApplicationResources" />
<plug-in className="net.sf.portions.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-
INF/validation.xml"/>
</plug-in>
</portlet-config>

```

In the portlet-config.xml configuration file there are five different sections:

- **"form-beans" element:** lists the different forms of the application. For each form it will be defined a "form-bean" element with the following attributes:
 - "name": id. of the form
 - "type": the class that implements the form
- **"global-forwards" element:** identifies the global redirections (default forwards) of the application. For each redirection it will be defined a "forward" element with the following attributes:
 - "name": id. of the forward
 - "mode": mode of the portlet (edit, help or view)
 - "path": url of the JSP page to which the redirection will be made
 - "content": content-type generated
- **"action-mappings" element:** maps each action of the application using "action" elements, that would have the following parameters:
 - "name": id. of the action

- "type": the class that implements the action
- "input": url of the JSP page to which the redirection, when there are errors while executing the plug-ins, will be made
- "content": content-type of the JSP page of the "input" parameter
- "formName": name (id.) of the form associated to the action

In addition, each "action" element will have as "forward" elements (as the ones defined in the "global-forwards" section) as possible redirections of the action.

- **"message-resources" element:** through the "parameter" attribute it specifies the properties file with the resources and messages of the application.
- **list of "plug-in" elements:** allows to list the plug-ins (for example, the validation plug-in) that must be executed by the controller. For each plug-in there will be a "plug-in" element with the following parameter:
 - "className": the class that implements the plug-in

If it is necessary it will be possible to specify configuration properties for each plug-in using the "set-property" sub-elements. Those elements have the following parameters:

- "property": id. of the property
- "value": value for the property

The PortletController and ConfigHelper classes

The controller, in the MVC pattern, has to process the request and to create any bean/object used by the JSP pages, and it has to decide, depending on the user actions, the JSP page to which the output must be redirected. Those tasks, in the PORTIONS framework, are done by the PortletController class.

The ConfigHelper class allows you to get the configuration data of the application that are indicated through the "portlet-config.xml" file.

PortletController

Properties

log

Log of the class

Methods

init(PortletConfig)

It performs the initialization tasks of the controller, which includes to realize a call to the init method of the ConfigHelper class to load the configuration data of the /WEB-INF/portlet-config.xml file.

doView(Forward, RenderRequest, RenderResponse)

It renders the portlet's view mode.

`doEdit(Forward, RenderRequest, RenderResponse)`

It renders the portlet's edit mode.

`doHelp(Forward, RenderRequest, RenderResponse)`

It renders the portlet's help mode.

`processAction(ActionRequest, ActionResponse)`

It process the portlet's action request.

`getActualForward(PortletRequest):Forward`

It allows to get the actual forward to which the controller must give the control.

ConfigHelper

Methods

`init(URL)`

It loads the configuration data from the URL to the portlet-config.xml file.

`getAction(String):Action`

It returns the configuration data of the action whose name has been passed as a parameter.

`getActionType(String):String`

It returns the value of the type property of the action whose name has been passed as a parameter.

`getActionInput(String):String`

It returns the value of the input property of the action whose name has been passed as a parameter.

`getActionContent(String):String`

It returns the value of the content property of the action whose name has been passed as a parameter.

`getActionScope(String):String`

It returns the value of the scope property of the action whose name has been passed as a parameter.

`getActionFormName(String):String`

It returns the value of the `formName` property of the action whose name has been passed as a parameter.

`getActionParameter(String):String`

It returns the value of the `parameter` property of the action whose name has been passed as a parameter.

`getActionForwards(String):Map`

It returns an instance of the class `Map` which contains the forwards associated to the action whose name has been passed as a parameter.

`getForward(String, String, String):Forward`

It returns the configuration data of the forward identified with the given parameters.

`getForwardPath(String, String, String):String`

It returns the value of the `path` property of the forward identified with the given parameters.

`getForwardContent(String, String, String):String`

It returns the value of the `content` property of the forward identified with the given parameters.

`getForm(String):FormBean`

It returns the configuration data of the form whose name has been passed as a parameter.

`getFormType(String):String`

It returns the value of the `type` property of the form whose name has been passed as a parameter.

`getDefaultForward(String, String):Forward`

It returns the configuration data of a global forward identified with the given parameters.

`getDefaultForwardPath(String, String):String`

It returns the value of the `path` property of a global forward identified with the given parameters.

getDefaultForwardContent(String, String):String

It returns the value of the content property of a global forward identified with the given parameters.

getMessageResources():String

It returns the configuration data that contains the path to the MessageResources file.

getPluginNames():List

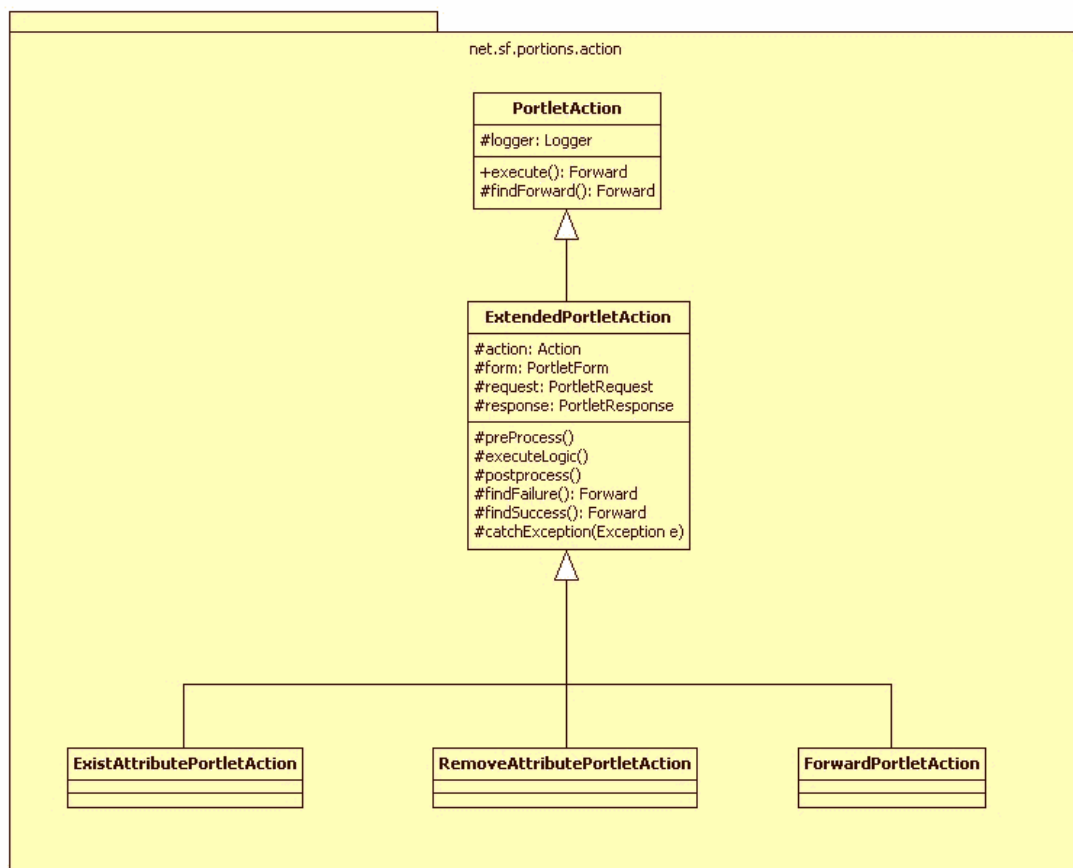
It returns a list with the names of the configured plug-ins.

getPluginParam(String, String):String

It returns the value of the parameter of the plug-in identified with the given parameters.

Actions hierarchy

Next, a Class Diagram that represents the Actions hierarchy given by the PORTIONS framework is showed:



- Illustration 1. Actions hierarchy of the PORTIONS framework -

PortletAction

Top hierarchy class for the Actions given by the PORTIONS framework. It is an abstract class that gives the basic functionality that may use the Actions.

Properties

log

Log of the class.

Methods

execute(Action , PortletForm , PortletRequest , PortletResponse):Forward

Abstract method that concrete Actions must rewrite to execute their tasks.

This method will be called by the application controller.

findForward(Action , String , String): Forward

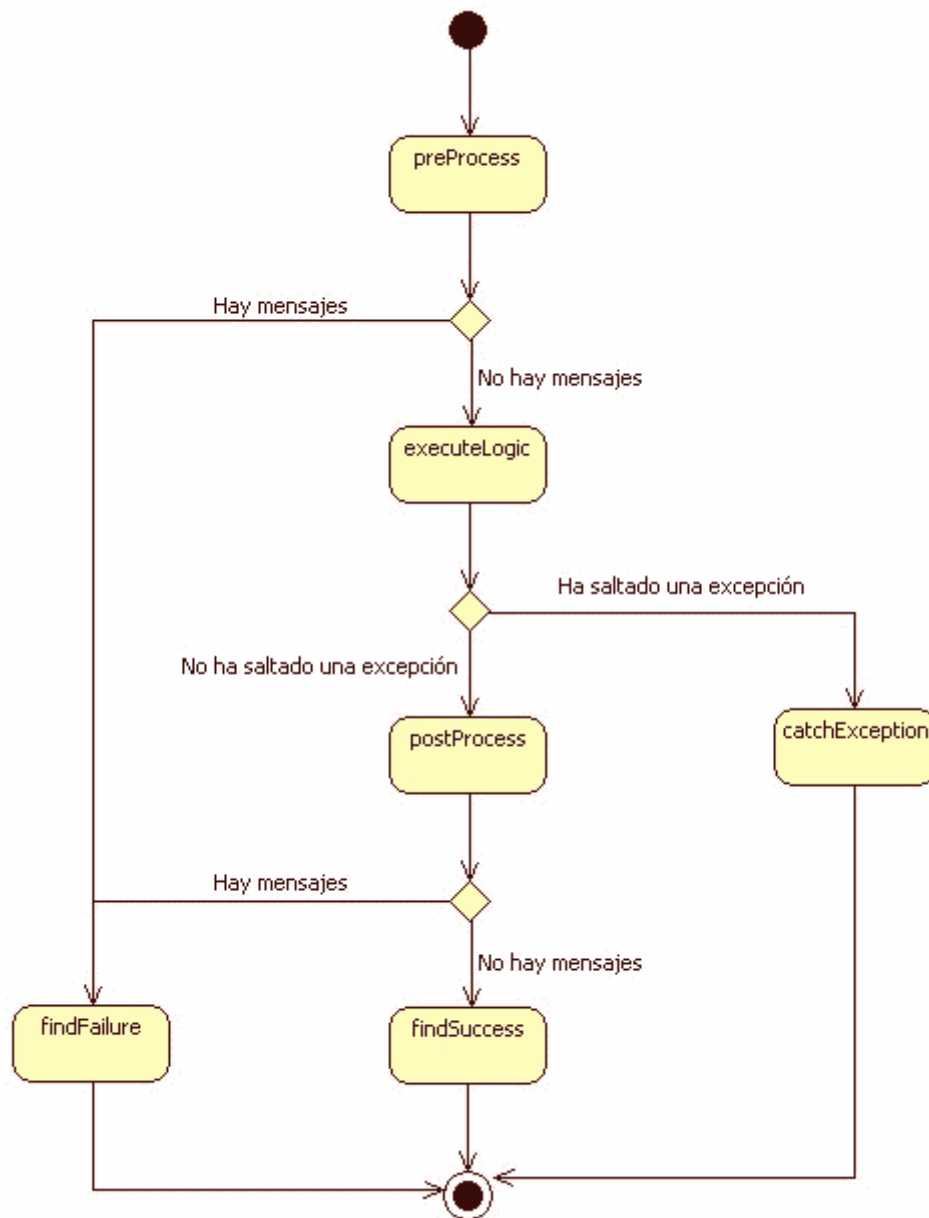
It allows to get the forward to which the application flow will be redirected.

The default implementation takes the forward of the instance of the Action class that is given as a parameter, and if it does not exist, looks for a global forward.

The subclasses can rewrite this method.

ExtendedPortletAction

Abstract class that represents an extension of the PortletAction. It implements the following State Machine:



- Illustration 2. State Machine of the ExtendedPortletAction class -

Properties

action

Instance of the `net.sf.portions.controller.configuration.Action` class with the configuration data of the action.

form

Instance of the `net.sf.portions.form.PortletForm` class that optionally (it depends on the configuration) can contains the parameters of the request.

request

The actual request (instance of the `javax.portlet.PortletRequest` class).

response

The generated response (instance of the `javax.portlet.PortletResponse` class).

Methods

`preProcess()`

It verifies the preconditions that the Action must fulfill.

If the preconditions are not fulfilled a message must be created. This creation of the message will redirect the execution flow to an error page as the method `findFailure()` will be invoked.

`executeLogic()`

It executes the business logic of the Action.

This method must be rewritten por those Actions that extend the `ExtendedPortletAction` class.

`postProcess()`

It verifies the postconditions that the Action must fulfill.

If the postconditions are not fulfilled a message must be created. This creation of the message will redirect the execution flow to an error page as the method `findFailure()` will be invoked.

`findFailure():Forward`

It retutns an instance of the `Forward` class that redirects to an error page.

The default implementation will get the "failure" `Forward` of the corresponding mode (view, edit or help).

`findSuccess():Forward`

It retutns an instance of the `Forward` class that redirects to a success page.

The default implementation will get the "success" `Forward` of the corresponding mode (view, edit or help).

`catchException(Exception)`

It must catch any exception thrown during the execution of the business logic of the Action.

The default implementation throws an `ExecuteActionException` which message is the message of the caught exception.

ExistAttributePortletAction

It verifies the existence of an attribute in some of the portlet's possible scopes (request or session). If the attribute exists, the execution flow will be passed to the "success" forward and, if it does not exist, to the "failure" one.

In the "parameter" property of the corresponding `<action-mapping>` it will be indicated the scope and the name of the attribute to look for. They will be separated by a ";" (semicolon):

```
parameter="application;HOURS"
```

If to look for the attribute in any of the possible scopes is wanted, a "*" (asterisk) must be used:

```
parameter="*;HOURS"
```

RemoveAttributePortletAction

It removes an attribute in some of the portlet's possible scopes (request or session). If the attribute exists and can be removed, the execution flow will be passed to the "success" forward and, if it does not exist or it can not be removed, to the "failure" one.

In the "parameter" property of the corresponding `<action-mapping>` it will be indicated the scope and the name of the attribute to remove. They will be separated by a ";" (semicolon):

```
parameter="application;HOURS"
```

If to remove the attribute in any of the possible scopes is wanted (the attribute will only be removed from the first scope in which it will be found), a "*" (asterisk) must be used:

```
parameter="*;HOURS"
```

ForwardPortletAction

It allows going from a JSP page to another by following the normal flow of the architecture (The JSP Model 2 Architecture for the JSR-168 portlets). To perform it, it is only necessary to configure the corresponding `<action-mapping>` with a "success" forward to the destination JSP page.

Forms

The "Forms" are javaBeans that optionally can be associated to a request. If it is configured in the `portlet-config.xml` configuration file, this bean will have its properties initiated with the corresponding parameters of the request before the associated Action to this request will be executed.

PortletForm

Top hierarchy class for the Forms given by the PORTIONS framework. It is an abstract class with the basic functionality that will be used by the Forms.

Properties

log

Log of the class.

Methods

validate()

It does the validations of the data loaded in its properties.

Right now it is not used by the controller, its use is not recommended.

reset()

It resets the value of the properties of the Form.

set(ValueObject)

It sets the value of the properties of the actual instance with the data contained in a ValueObject.

populate():ValueObject

It returns a ValueObject loaded with the data of the actual instance.

View

"portions" tag library

messages

<portions:messages>: it shows the messages generated by the application (normally the messages will be validation errors, or errors produced during the execution).

Attributes

property

Property to which the messages are associated. By default, all messages will be shown, without distinguishing the property to which they are associated.

id

Identifies the values of the header, prefix, suffix and footer to use while showing the messages. By default, they will be shown those corresponding to the "portions.messages.header", "portions.messages.prefix", "portions.messages.suffix" and "portions.messages.footer" keys of the MessageResources file. If an "id" is specified, to those keys it will be added a "." (point) followed by the id given.

```
<%@ taglib uri='http://java.sun.com/portlet' prefix='portlet'%>
<%@ taglib uri="/WEB-INF/tld/portions.tld" prefix="portions" %>
<portlet:defineObjects/>
<portions:messages />
```

getResource

<portions:getResource>: it gets the value associated to a key in the MessageResources file.

Attributes

key

key of the MessageResources file to get.

```
<%@ taglib uri='http://java.sun.com/portlet' prefix='portlet'%>
<%@ taglib uri="/WEB-INF/tld/portions.tld" prefix="portions" %>
<portlet:defineObjects/>
<portions:getResource key="label.helloWorld" />
```

redirect

<portions:redirect>: perform a automatic redirection from a JSP to an action of the portlet.

Attributes

id

Identify the values of the id, the value and the css class of the button of the form that will perform the redirection. By default, they will be shown those corresponding to the "portions.redirect.formId", "portions.redirect.submitValue" and "portions.redirect.submitClass" keys of the MessageResources file. If an "id" is specified, to those keys it will be added a "." (point) followed by the id given.

action

It indicates the name of the action to execute.

mode

It says the portlet mode to show (help, edit or view).

```
<%@ taglib uri='http://java.sun.com/portlet' prefix='portlet'%>
```

```
<%@ taglib uri="/WEB-INF/tld/portions.tld" prefix="portions" %>
<portlet:defineObjects/>
<portions:redirect action="confirmAction" mode="view" />
```

encodeURL

<portions:encodeURL>: it encodes the URL of a static resource (image, document, ...) included in the WAR file of the portlet.

Attributes

url

url to encode.

```
<%@ taglib uri='http://java.sun.com/portlet' prefix='portlet' %>
<%@ taglib uri="/WEB-INF/tld/portions.tld" prefix="portions" %>
<portlet:defineObjects/>
"
title="Sample image">
```

Form validation

PORTIONS framework includes a form validation system (based in the Commons Validator project) for which is necessary to define some validation rules and to map the form fields against those rules in one or some XML files (they use to be the validation-rules.xml and validation.xml files, or simply the validation.xml file).

Next, a validation.xml sample file is shown:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules
    Configuration 1.3.0//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">

<form-validation>
  <global>
    <validator name="required"
      classname="net.sf.portions.validator.RequiredValidator"
      "
      method="validateRequired"
      methodParams="java.lang.Object,
org.apache.commons.validator.Field"
      msg="error.required"/>

    <validator name="mustBeSelected"
      classname="net.sf.portions.validator.RequiredValidator"
      "
      method="validateRequired"
      methodParams="java.lang.Object,
org.apache.commons.validator.Field"
      msg="error.must_select"/>
  </global>

  <formset>
    <form name="patternForm">
```

```

        <field property="pattern" depends="required">
            <arg key="patterForm.patter"/>
        </field>
    </form>

    <form name="idForm">
        <field property="id" depends="mustBeSelected">
            <arg key="idForm.id"/>
        </field>
    </form>
</formset>
</form-validation>

```

In the example you can see how the validation rules and the mapping have been combined in a single file:

- The validation rules are into the "global" element. As it can be observed, there have been defined two different validation rules using the same validator, one called "required" and the other "mustBeSelected".
- The mapping between the form fields and the validation rules are specified into the "formset" element. Each form must be mapped into a "form" element that must have those validations that must be performed on each field in the "field" elements.

You can learn more about form validations in the [Commons Validator Web site](#).

Validators included in the PORTIONS framework


required

Class: net.sf.portions.validator.RequiredValidator

Method: validateRequired

It validates the existence of a field in the form and it also validates that this field is not empty.

Pagination Tables

To create pagination tables you can use the DisplayTag library (<http://displaytag.sourceforge.net/>) . This library allows to create a HTML table with pagination and order capabilities being started off of a data collection using a set of personalized tags.

To be able to use the DisplayTag library with the PORTIONS framework, this last one includes some classes that must be configured in the displaytag.properties file as follow:

```
factory.requestHelper=net.sf.portions.displaytag.PortletRequestHelperFactory
```

Model

Data Objects

ValueObject (Interface)

The PORTIONS framework gives the "ValueObject" interface that the data objects must implement (Value Object pattern). This interface defines the toXML() method that allows to get a description of the object in XML format.

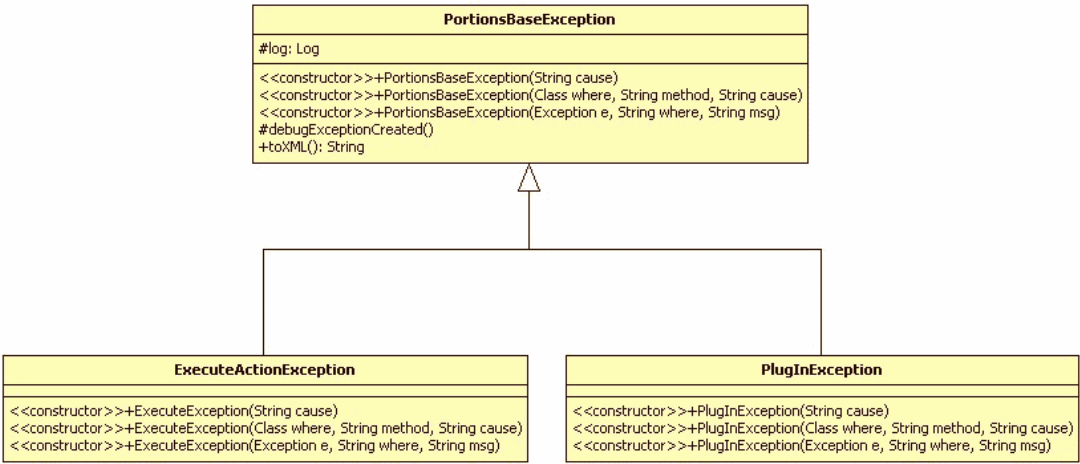
BaseVO

To help with the implementation of the data objects, the PORTIONS framework includes the "BaseVO" class which implements the toXML() method defined in the ValueObject interface.

Other

Exceptions hierarchy

Next, a Class Diagram that represents the Exceptions hierarchy given by the PORTIONS framework is showed:



- Illustration 3. Exceptions hierarchy of the PORTIONS framework -

PortionsBaseException

Top hierarchy class for the Exceptions given by the PORTIONS framework. It is a class that gives the basic functionality that may use the Exceptions of the framework.

Properties

log

The log of the class.

Methods

<<constructor>>PortionsBaseException(String)

Constructor for the exception that gets as a parameter the description of the cause or the place where the exception has happened.

<<constructor>>PortionsBaseException(Class , String , String)

Constructor for the exception that gets as parameters the class where it is produced, the name of the method where where it is thrown and the description of the cause of the exception.

<<constructor>>PortionsBaseException(Exception, String, String)

Constructor for the exception that gets as parameters the original exception (it will be anidated), the place where the exception is thrown and the cause.

debugExceptionCreated()

Writes debug information about the exception in the log.

toXML():String

Gives a XML representation of the exception.

ExecuteActionException

Exception that must be thrown by an action if, while it is executed, an error or exception situation happened.

If an ExecuteActionException reaches the controller, this will show an error message in the log of the portlet.

PlugInException

Exception that must be thrown by a plug-in (class that implements the net.sf.portions.plugin.IPortletPlugIn interface) if, while it is executed, an error or exception situation happened.

The controller, when captures an exception of this type, will get the messages generated while the plug-in execution and will redirect the application flow to the JSP page indicated in the "input" parameter of the actual action.

Messages

The messages is used for notifying the user the cause of an error (by example the validation errors) or the especial situations produced while the Actions are executed.

The PORTIONS framework has different classes to treat the messages (and the tag <portions:messages> explained in the [View](#) section of this document):

MessageHelper

Utility class that has some methods to perform common tasks being related with the messages generated by the application.

Methods

`saveMessages(PortletRequest, PortletMessages)`

Saves the messages given as parameters in the session context.

`setMessagesInRequest(PortletRequest)`

Takes the messages from the session context (and deletes them in this context) and saves them in the request context.

`thereAreMessages(PortletRequest)`

Returns 'true' if there are messages in the session context.

PortletMessage

Class that represents a message generated by the application. It represents the key to a message included into the MessageResources file of the application, and optionally the values to replace the parameters {0}, {1}, ... {n} of the message.

Properties

key

Key of the message in the MessageResources file (available through the `getKey():String` method).

values

Replacement values for the message (available through the `getValues():Object[]` method).

Methods

`<<constructor>>PortletMessage(String)`

Constructor of the message that receives as parameters the key of the MessageResources file.

`<<constructor>>PortletMessage(String , Object[])`

Constructor of the message that receives as parameters the key of the MessageResources file and the replacement values.

`getKey():String`

Returns the value of the key property.

getValues():Object[]

Returns the value of the values property.

PortletMessages

Messages collection that allows a hierarchical structuring of messages.

Properties

GLOBAL_MESSAGE

Name of the property to save the global messages (those which are not related with a particular property).

Methods

<<constructor>>PortletMessages()

Constructor without parameters for the class.

<<constructor>>PortletMessages(PortletMessages)

Constructor for the class that adds to the new instance the messages that are contained in another instance given as a parameter.

add(String, PortletMessage)

Adds a message to the collection of messages associated to the given property (if the property is null or is empty, it will be added to the PortletMessages.GLOBAL_MESSAGE property).

add(PortletMessages)

Adds the messages contained in the instance given as a parameter to the actual collection of messages.

clear()

Deletes all the messages that the actual instance contains.

isEmpty()

Returns 'true' if the collection of messages is empty.

get():Iterator

Returns the collection of messages as an Iterator.

get(String):Iterator

Returns the collection of messages associated to the specified property (if the property is null or is empty, it will be added to the PortletMessages.GLOBAL_MESSAGE property).

properties():Iterator

Returns an Iterator with the properties used to save the messages.

size():int

Returns the number of property-key pairs added.

size(String):int

Returns the number of messages associated to the specified property (if the property is null or is empty, it will be added to the PortletMessages.GLOBAL_MESSAGE property).