Developer's Guide

(Work in progress)

Table of Contents

Chapter 1. Setting up the development environment

If you want to start developing with WebOnSwing you must configure some parameters to your servlet container and get some useful files. To run your application you can use any compatible Java Servlet 2.2 container as Apache Tomcat or Jetty. Once the server is installed and fully functional, you have to add **WebOnSwingBoot.jar** to its **bootclasspath**, and then you will be able to deploy any WebOnSwing application. In order to simplify the first steps in development you can use the war file **WebOnSwingEmptyWebApp.war** (build it running the ant task in source distribution), that contains the minimal set of files to start a WebOnSwing application. So you only have to add the application specific window classes, templates files and resources.

1.1. WebOnSwing Configuration

WebOnSwing is configured througth "webonswing-framework.config.xml" file.

In this file you could set the path of other configuration files, such as template engine, page manager, contributor manager and the window tree state (deprecated).

You could specify your own component name manager to provide other method to resolve components names, for example assigning the java reference as its name (this could be used only for statefull applications).

WindowManager and Hierarchy wrapper work together to performs the component hierarchy adaptation (in this case is for Swing).

Resources path is the default place of web files like .js, .html, .gif, .jpg, etc.

The "compress-persisted-data" element tells the framework if window manager state and persitence data of each component will be compressed or not in html page, the compressing operation could slow down your page responding so use it if you want to shrink your page and have enough CPU at server.

And with catch exceptions activated the framework will print the stack trace in a special page and in case it's disable the exception will be catched by the servlet mechanism to perform a desired operation depending of the exception type.

```
<webonswing-framework>
        <page-manager-config-xml-file>/net/ar/webonswing/config/page-manager.config.xml</page-manager-config-xml-file>
        <template-manager-config-xml-file>/net/ar/webonswing/config/template-manager.config.xml</template-manager-config-xml-file>
        <contributor-manager-config-xml-file>/net/ar/webonswing/config/contributor-manager.config.xml</contributor-manager-config-xm
        <window-tree-state-manager-config-serialized-file>/net/ar/webonswing/config/window-tree-state-manager.config.serialized</wir

        <component-name-manager-class-name>net.ar.webonswing.managers.names.CharacterComponentNameManager</component-name-manager-cl

        <window-manager-class-name>net.ar.webonswing.wrapping.swing.SwingWindowManager</window-manager-class-name>
        <hierarchy-wrapper-class-name>net.ar.webonswing.wrapping.swing.SwingHierarchyWrapper</hierarchy-wrapper-class-name>

        <resources-path>/net/ar/webonswing/resources</resources-path>
        <compress-persisted-data>false</compress-persisted-data>
        <catch-exceptions>true</catch-exceptions>
</webonswing-framework>
```

Chapter 2. Hello World example

This first example shows how to create a simple 'hello world' application, that uses a JDialog as main container and adds a JLabel component to its content pane. WebOnSwing can display pages based in any class that extends java.awt.Window and implements the RootPaneContainer interface, these are **JFrame, JDialog and JWindow**. You may access to this page browsing the URL:
http://localhost/net/ar/webonswing/tutorial/HelloWorld.page

```
package net.ar.webonswing.tutorial;

import javax.swing.*;

public class HelloWorld extends JDialog
{
  public HelloWorld()
  {
    getContentPane().add(new JLabel("Hello World!"));
  }
}
```

Chapter 3. Event handling

We are going to show how to work with events and listeners building a simple calculator application. You can also take a look at Swing lessons that covers listener and event treatment at the following URL: http://java.sun.com/docs/books/tutorial/uiswing/events/index.html

The calculator application has two text field, one combo box with available operations, a result label and a process button to perform the desired calculus. This class implements an ActionListener interface to handle events fired by clicking the button. When this button is pressed the operation to perfrom is taken from the selected item of JComboBox, the two numbers from input fields and the result is displayed in resultLabel. Note that every component state is updated before click event is fired, so when is time to process this event all data in window reflects the real state.

You may access to this page browsing the URL:

http://localhost/net/ar/webonswing/tutorial/Calculator.page

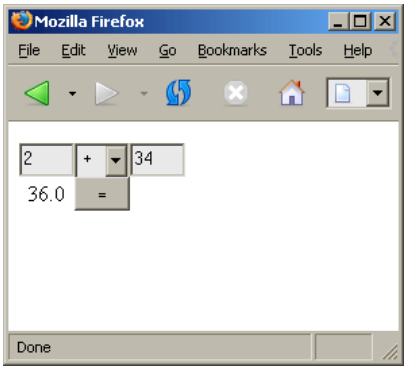

Figure 3.1. Calculator screenshot

```
package net.ar.webonswing.tutorial;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class Calculator extends JDialog implements ActionListener
{
  protected JLabel resultLabel;
  protected JTextField firstNumerField;
  protected JTextField secondNumerField;
  protected JComboBox operationCombo;

  public Calculator()
  {
    Container contentPane= getContentPane();

    resultLabel= new JLabel("result");
    firstNumerField= new JTextField("");
    secondNumerField= new JTextField("");
    operationCombo= new JComboBox(new String[] { "+", "-", "*", "/"});
    JButton processButton= new JButton("=");

    processButton.addActionListener(this);

    contentPane.setLayout(new GridLayout(2, 3));
    contentPane.add(firstNumerField);
    contentPane.add(operationCombo);
    contentPane.add(secondNumerField);
    contentPane.add(resultLabel);
    contentPane.add(processButton);

    pack();
  }

  public void actionPerformed(ActionEvent e)
  {
```

```
    float firstNumber= Float.parseFloat(firstNumerField.getText());
    float secondNumber= Float.parseFloat(secondNumerField.getText());
    float result= 0;

    switch (operationCombo.getSelectedItem().toString().charAt(0))
    {
      case '+':
          result= firstNumber + secondNumber;
          break;
      case '-':
          result= firstNumber - secondNumber;
          break;
      case '*':
          result= firstNumber * secondNumber;
          break;
      case '/':
          result= firstNumber / secondNumber;
          break;
    }

    resultLabel.setText(Float.toString(result));
  }
}
```

Chapter 4. WebOnSwing Template Engine

4.1. Template Engine Configuration

If you want to work with new templates you can add them to '/net/ar/webonswing/html' package, and in case you want organize them in other classpath you must edit the template manager configuration file for adding the new resources path there.

Template manager configuration (template-manager.config.xml)

```
<template-manager>

  <template-cache-active>true</template-cache-active>

  <resources-paths>
    <path>/net/ar/webonswing/resources/html</path>
    <path>/net/ar/webonswing/swing/components</path>
  </resources-paths>

  <template-aliases>
    <template>
      <alias>RefreshComponentsMain</alias>
      <resource-path>/net/ar/webonswing/resources/html/UpdateComponentsMain.html</resource-path>
    </template>
  </template-aliases>

</template-manager>
```

4.2. HTML Templates

WebOnSwing provides a powerful template engine, that lets you apply html designs to your applications. This html template files dont need an special syntax, you just have to add a "name" attribute to the tags you want to turn into a placeholder or a subtemplate. The template engine will parse this files creating a template model with nested subtemplates, and you will be able to traverse this model with a very simple path syntax. For example, if you have the following template:

```
<table name="template1">
 <tr name="template2">
  <td name="template3">
   hello
  </td>
 </tr>
 <tr name="template4">
  <td name="template5">
   world
  </td>
 </tr>
</table>
```

The path to locate the "template3" within the whole "template1" would be "template1.template2.template3", getting the subtemplate '<td>hello</td>'.

This polimorphic behavior lets you interchange html templates transparently, for example to create skins, because there is no logic inside the files, it's just the view (a pure html file). When you need to repeat some parts of a template in a page, you can get the part (subtemplate) accessing via its path, and then apply this subtemplate to the desired containers or components by a TemplateLayout. In case you want to perform this operation automatically, you can use the layout manager PropagateTemplateLayoutByName, that will replace every GridLayout (which does not specify the rows or columns count) with a TemplateFlowLayout, assigning the desired subtemplate to each subcomponent.

4.3. Applying templates with new layout managers

We will see how to change the view of the calculator application using templates, acquiring a professional look to your web application instead of placing components by coordinates. Some changes in source file will be done, replacing the GridLayout by TemplateLayout and using some contraints when each component is added to the container.

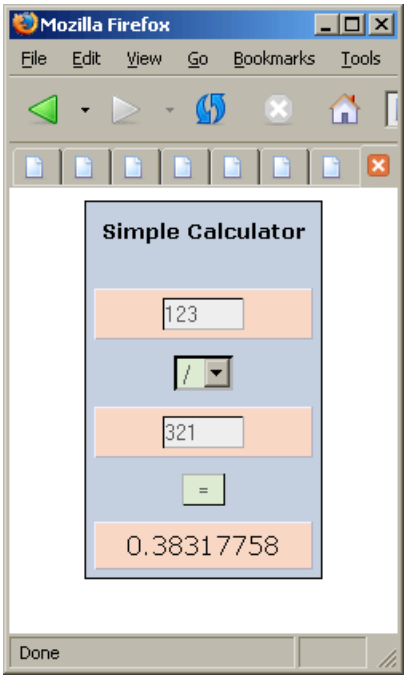

Figure 4.1. Calculator with template screenshot

```
package net.ar.webonswing.tutorial;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import net.ar.webonswing.*;
import net.ar.webonswing.swing.layouts.*;

public class Calculator extends JDialog implements ActionListener
{
  protected JLabel resultLabel;
  protected JTextField firstNumerField;
  protected JTextField secondNumerField;
  protected JComboBox operationCombo;

  public Calculator()
  {
    Container contentPane= getContentPane();

    resultLabel= new JLabel("result");
    firstNumerField= new JTextField("");
    secondNumerField= new JTextField("");
    operationCombo= new JComboBox(new String[] { "+", "-", "*", "/"});
    JButton processButton= new JButton("=");

    processButton.addActionListener(this);

    contentPane.setLayout(new TemplateLayout(WosFramework.getKeyPositionTemplateForName("CalculatorTemplate")));

    contentPane.add(firstNumerField, "firstNumber");
    contentPane.add(operationCombo, "operation");
    contentPane.add(secondNumerField, "secondNumber");
    contentPane.add(resultLabel, "result");
    contentPane.add(processButton, "button");
  }

  public void actionPerformed(ActionEvent e)
  {
```

```
    float firstNumber= Float.parseFloat(firstNumerField.getText());
    float secondNumber= Float.parseFloat(secondNumerField.getText());
    float result= 0;

    switch (operationCombo.getSelectedItem().toString().charAt(0))
    {
      case '+':
          result= firstNumber + secondNumber;
          break;
      case '-':
          result= firstNumber - secondNumber;
          break;
      case '*':
          result= firstNumber * secondNumber;
          break;
      case '/':
          result= firstNumber / secondNumber;
          break;
    }

    resultLabel.setText(Float.toString(result));
  }
}
```

And for example we can use the following html file.

CalculatorTemplate.html

```
<table cellspacing="5" cellpadding="5" border="0" align="center" style="font-family : Verdana;
  background-color: #C4D0DF;border: 1px;border-style: solid;border-color: #000000;">
  <tr>
    <td align="center"><font face="Verdana" size="2px"><b>Simple Calculator</b></font><br><br></td>
  </tr>
  <tr>
    <td align="center" style="border: 1px;border-style: solid;border-color: #F0F0FF #B0B0C0 #B0B0C0 #F0F0FF;
      background-color: #F8D8C4;">
      <span name="firstNumber" size="5" style="color:#555555; border-color: #000000; border-width: 1px;
        background-color: #EEEEEE;"/>
    </td>
  </tr>
  <tr>
    <td align="center">
      <span name="operation" style="color:#555555; border-color: #000000; border-width: 1px; background-color:
        #DCEBD1;"/>
    </td>
  </tr>
  <tr>
    <td align="center" style="border: 1px;border-style: solid;border-color: #F0F0FF #B0B0C0 #B0B0C0 #F0F0FF;
      background-color: #F8D8C4;">
      <span name="secondNumber" size="5" style="color:#555555; border-color: #000000; border-width: 1px;
        background-color: #EEEEEE;"/>
    </td>
  </tr>
  <tr>
    <td align="center">
      <span name="button" style="color:#555555; border-color: #000000; border-width: 1px;
        background-color: #DCEBD1;"/>
    </td>
  </tr>
  <tr>
    <td align="center" style="border: 1px;border-style: solid;border-color: #F0F0FF #B0B0C0 #B0B0C0 #F0F0FF;
      background-color: #F8D8C4;">
      <span name="result" />
    </td>
  </tr>
</table>
```

4.4. Applying templates quickly

With PropagateTemplateLayoutByName layout manager you can apply a template and its subtemplates to the whole component hierarchy. You have to set this layout manager to the root component of the hierarchy you want to convert, constructing it with the alias of the desired html template.

Once the layout process is started, PropagateTemplateLayoutByName will traverse the component hierarchy reading each component name property and trying to match this with the name of a subtemplate (or

placeholder) in current template level.

This process is performed recursively and for each component/template pair this layout will set a new PropagateTemplateLayoutByName instance to the found component, constructed with the matching template. In case the found component has a GridLayout manager with either the row count or col count set with a value of 0, this layout will be replaced by a TemplateFlowLayout that will produce a concatenation of each subcomponent rendering. These subcomponents also could recieve a propagation of templates if each component name match with a template in current level and so on.

To understand better this concept consider the following example:

```
package net.ar.webonswing.tutorial;

import javax.swing.*;

import net.ar.webonswing.*;

public class PropagateTemplatesExample extends JDialog
{
  public PropagateTemplatesExample()
  {
    JLabel label= new JLabel("label");
    JPanel panel1= new JPanel();
    JButton button= new JButton("button");
    JPanel panel2= new JPanel();
    JCheckBox checkButton= new JCheckBox("check button");

    label.setName("label");
    button.setName("button");
    checkButton.setName("checkButton");
    panel1.setName("panel1");
    panel2.setName("panel2");

    panel1.add(button);
    panel1.add(panel2);
    panel2.add(checkButton);
    getContentPane().add(label);
    getContentPane().add(panel1);

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor("PropagateTemplatesExample1"));
  }
}
```

And, for example, we may use the following templates:

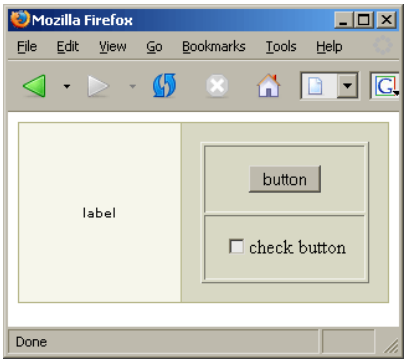

Figure 4.2. PropagateTemplatesExample screenshot 1

PropagateTemplatesExample1.html

```
<table border="0" cellspacing="0" cellpadding="0" bgcolor="#B7B78B">
  <tr>
    <td>
      <table border=0" cellspacing="1" cellpadding="15">
        <tr>
          <td bgcolor="#F6F6EB" width="100" align="center">
            <font face="Verdana" size="-2">
              <span name="label"/>
```

```
          </font>
        </td>
        <td bgcolor=#D8D8C4>
          <table border="1" name="panel1" cellpadding="15">
            <tr>
              <td align="center"><span name="button"/></td>
            </tr>
            <tr name="panel2">
              <td><span name="checkButton"/></td>
            </tr>
          </table>
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

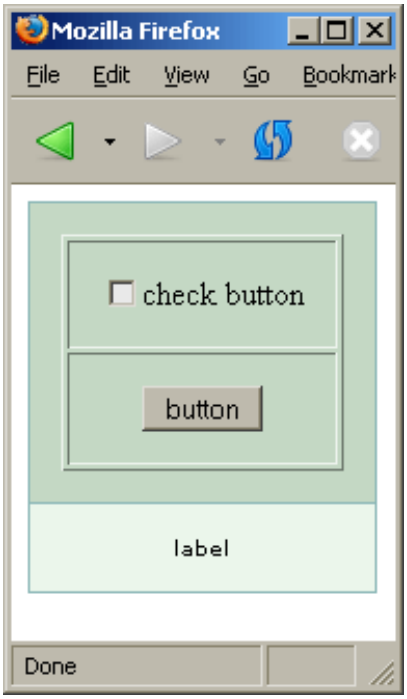

Figure 4.3. PropagateTemplatesExample screenshot 2

PropagateTemplatesExample2.html

```
<table border="0" cellspacing="0" cellpadding="0" bgcolor="#8BB7B7">
  <tr>
    <td>
      <table border=0" cellspacing="1" cellpadding="15">
        <tr>
          <td bgcolor=#C4D8C4>
            <table border="1" name="panel1" cellpadding="15">
              <tr name="panel2">
                <td><span name="checkButton"/></td>
              </tr>
              <tr>
                <td align="center"><span name="button"/></td>
              </tr>
            </table>
            </td>
          </tr>
        <tr>
          <td bgcolor="#EBF6EB" width="100" align="center">
            <font face="Verdana" size="-2">
              <span name="label"/>
            </font>
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

Note that this last template change the internal representation of the JCheckBox because in this case the "checkButton" tag is a subtemplate (not only a placeholder), and CheckBoxUIContributor accepts templates with this structure.

Chapter 5. Page navigation

Page navigation is performed in a similar way you open windows in classic Swing applications. To open or close windows, modal or not, you have to call the static methods provided by the framework, instead of using the method Component.setvisible(boolean). These methods are:

- WosFramework.showChildWindow(parentWindow, childWindow)
- WosFramework.showAndExecute(parentWindow, childWindow, "method name")
- WosFramework.hide(windowToClose)

In showAndExecute method you have to provide a method name of the parent window that recieve a childWindow instance as parameter. This method will be called when the child window is closed and the parent window needs to process some data of the closed one.

5.1. Page Manager Configuration

Page manager is configured througth "page-manager.config.xml" file.

You can set there the default page class for every window that dont have an explicit page assigned, in case you want to do some specific operation over certain page event you may specify your own page class.

The window place parameter defines if the page will be statefull or stateless, the accepted values are "session" or "page". With "session" option the Window Manager will be placed in session and all openned windows will be stored there and with "page" option the Window Manager is "serialized" and stored in html page. Also each window/page could stablish its own policy to treat states and some could be stored in session and others in page, for example for diferent parts of the same site.

And the element pages, lets you define an arbitrary web path for a window, its page and the place it will be stored.

```
<page-manager>
  <default-window-place>page</default-window-place>
  <default-page-class>net.ar.webonswing.pages.SaveContextDefaultHtmlPage</default-page-class>
  <pages>
    <web-page>
      <page-class>net.ar.webonswing.pages.ExternalHtmlPage</page-class>
      <web-path>/WosExamples/WebPage</web-path>
      <window-class>net.ar.webonswing.swing.components.WebPage</window-class>
      <window-place>page</window-place>
    </web-page>
  </pages>
</page-manager>
```

5.2. Default window manager behaviour

WebOnSwing DefaultWindowManager contains a collection of the opened windows. Its default behavior has the mision of keep the history of opened modal windows, so it use a stack of windows to store this 'navigation' in order to obtain or reconstruct each parent window when its child is closed. Every time you open a new modal window, the window manager will push it in the stack, and this window will become active and visible. This window will keep on top of the stack until 'hide' method is called, at this moment the window manager will pop it and turn the parent window active. In case you decide to open a non modal window, the window manager

will discard all windows in the stack and will push it alone.

Each window has an unique id to identify it inside the stack, this will be used by the window manager for example to change the active window if the user press browser back button and fire an event to a "parent page". First at all, window manager will search in the stack if exist any window with the posted id, and if it's found, will pop every window until leaving this one on top. This last procedure must be used when the window manager is place in web session (statefull), because it has to synchronize the stack state with the browser history.

But in almost any web application we need to use a stateless aproach, so WebOnSwing can store the window manager directly as a state in html page, with no need to synchronize the window stack with browser history because the stack is inside the page. As you dont need to store any information about navigation in session your application is completly stateless, and you can move throught any page with back and forward buttons, restoring for each case the corresponding window manager.

When the window manager is persisted in web page the stack of windows is preserved, but we can't store each window instance because it could be a very big amount of data, so each window instance is replaced by a VirtualWindow that consist of the window class name and the persisted state of this instance, being the same state used for restoring a window in "stateless mode" and is built using all persistence contributions of the window. This virtual window uses a significant less information to represent a window instance, reducing the weight of the page and traffic.

5.3. JLink component

Besides those previously introduced static methods for opening and hiding pages, we can use the JLink component to include a link like behaviour to any UI. This component can be constructed with a ordinary URL string (external links) or using class of the window you want to open (webonswing pages).

5.4. Login example

When Login Button is pressed open a modal window with user and password fields. If the user enter the correct pair, this modal window is closed and "processLogin" method is called, passing the corresponding window instance.

```
public class LoginExample extends JDialog
{
  protected JLabel label= new JLabel("");

  public LoginExample()
  {
    JButton topSecret= new JButton("Login here!");
    topSecret.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent aE)
      {
        if (WosFramework.getSharedContext().get("user") == null)
        {
          Login loginWindow= new Login();
          loginWindow.setModal(true);
          WosFramework.showAndExecute(LoginExample.this, loginWindow, "processLogin");
        }
        else
          label.setText("Welcome again!!!");
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));
    getContentPane().add(label);
    getContentPane().add(topSecret);
  }

  public void processLogin(Login aLoginWindow)
  {
```

```
    label.setText("Welcome " + aLoginWindow.userField.getText() + "!!!");
  }
}
```

```
public class Login extends JDialog
{
  public JTextField userField= new JTextField ();

  public Login()
  {
    final JPasswordField passwordField= new JPasswordField();
    JButton button= new JButton("Login");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent aE)
      {
        if (userField.getText().equals("maradona") && passwordField.getText().equals("pelusa"))
          {
            WosFramework.getSharedContext().put("user", "diego");
            WosFramework.hide(Login.this);
          }
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));
    getContentPane().add(userField);
    getContentPane().add(passwordField);
    getContentPane().add(button);
  }
}
```

Chapter 6. Validation components

6.1. Introduction to Validation

WebOnSwing includes a set of validation components that provide an easy-to-use but powerful way to check input fields for errors and, if necessary, display messages to the user. Validation components are added to pages like other Swing components. There are validation components for specific types of validation, such as range checking or pattern matching, plus a JRequiredFieldValidator that ensures that a user does not skip an entry field. You can attach more than one validation component to an input component. For example, you might specify both that an entry is required and that it must contain a specific range of values. This components work with a limited subset of Swing input components. For each component, a specific property contains the value to be validated. The following table lists the input components that may be validated.

Table 6.1. Validated components properties

Swing Component	Validated Property
JTextField	text
JTextArea	text
JPasswordField	password
JList	selectedValue
JComboBox	selectedItem
JToggleButton	selected

6.2. Types of Validation Components

The simplest form of validation is a required field. If the user enters any value in a field, it is valid. The following example illustrates this using the JRequiredFieldValidator.

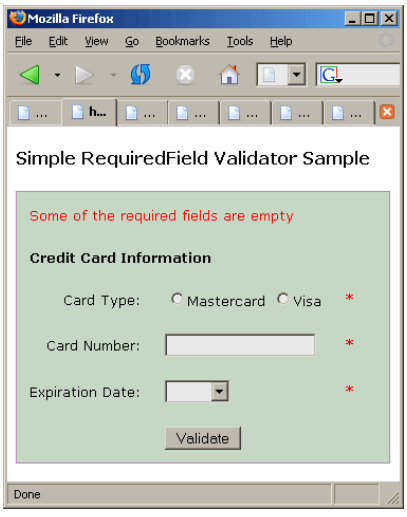

Figure 6.1. JRequiredFieldExample screenshot

```
public class JRequiredFieldExample extends JDialog
{
  public JRequiredFieldExample()
  {
    final JLabel label= new JLabel();

    JRadioButton card1= new JRadioButton("Mastercard");
    JRadioButton card2= new JRadioButton("Visa");
    ButtonGroup cardsGroup= new ButtonGroup();
    cardsGroup.add(card1);
    cardsGroup.add(card2);

    JTextField cardNumberField= new JTextField();

    JComboBox expirationCombo= new JComboBox(new String[] { "", "06/04", "07/04", "08/04", "09/04", "09/05", "10/05",
      "11/05", "12/05"});

    final JRequiredFieldValidator cardsRequiredFieldValidator= new JRequiredFieldValidator(card1, "*", false);

    final JRequiredFieldValidator expirationRequiredFieldValidator= new JRequiredFieldValidator(expirationCombo,
      "*", false);

    final JRequiredFieldValidator numberRequiredFieldValidator= new JRequiredFieldValidator(cardNumberField,
      "*", false);

    JButton button= new JButton("Validate");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        if (cardsRequiredFieldValidator.doValidation() & expirationRequiredFieldValidator.doValidation() &
    numberRequiredFieldValidator.doValidation())
          label.setText("Page is Valid!");
        else
          label.setText("Some of the required fields are empty");
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));

    getContentPane().add(label).setName("label1");
    getContentPane().add(card1).setName("card1");
    getContentPane().add(card2).setName("card2");
    getContentPane().add(cardsRequiredFieldValidator).setName("cardsValidator");
    getContentPane().add(cardNumberField).setName("number");
    getContentPane().add(numberRequiredFieldValidator).setName("numberValidator");
    getContentPane().add(expirationCombo).setName("expirationCombo");
    getContentPane().add(expirationRequiredFieldValidator).setName("expirationValidator");
    getContentPane().add(button).setName("button");

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor("JRequiredFieldExample.main"));
  }
}
```

JRequiredFieldExample.html

```
<html>
<head>
</head>

<body>
<span name="main">
<h4><font face="Verdana">Simple RequiredField Validator Sample</font></h4>
<p>
    <table bgcolor="#C4D8C4" cellpadding=10 style="border: 1px solid #B78BB7">
    <tr valign="top">
      <td colspan=3>
        <font face="Verdana" color="Red" size="2"><span name="label1"/></font>
        <br>
      </td>
    </tr>
    <tr>
      <td colspan=3>
      <font face=Verdana size=2><b>Credit Card Information</b></font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>Card Type:</font>
      </td>
      <td>
        <font face=Verdana size=2>
        <input name="card1" type="radio" />
        <span name="card2"/>
        </font>
      </td>
      <td align=middle rowspan=1>
```

```
    <font color="Red"><span name="cardsValidator"/></font>
  </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>Card Number:</font>
      </td>
      <td>
        <input type="text" name="number" />
      </td>
      <td>
        <font color="Red"><span name="numberValidator"/></font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>Expiration Date:</font>
      </td>
      <td>
        <span name="expirationCombo" />
      </td>
      <td>
        <font color="Red"><span name="expirationValidator"/></font>
      </td>
      <td>
    </tr>
    <tr>
      <td></td>
      <td>
        <input name="button" value="Validate" />
      </td>
      <td></td>
    </tr>
    </table>
</span>
</body>
</html>
```

There are also validation components for specific types of validation, such as range checking or pattern matching. The following table lists the validation components.

Table 6.2. Types of validators

Validation Component	Description
JRequiredFieldValidator	Ensures that the user does not skip an entry.
JCompareValidator	Compares a user's entry with a constant value or a property value of another component using a comparison operator (less than, equal to, greater than, and so on).
JRangeValidator	Checks that a user's entry is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, dates, or another two components. Boundaries can be expressed as constants.
JRegularExpressionValidator	Checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.
JGroupValidator	Displays the validation errors in summary form for all of the added validators.

6.3. Client-Side Validation

The validation components always perform validation checking in server code. However, if the user is working with a browser that supports DHTML, the validation components can also perform validation using client script. With client-side validation, any errors are detected on the client when the form is submitted to the server. If any of the validators are found to be in error, the submission of the form to the server is cancelled and the validator's message property is displayed. This permits the user to correct the input before submitting the form to the server. Field values are revalidated as soon as the field containing the error loses focus, thus providing the user with a rich, interactive validation experience.

Note that the page always performs validation on the server, even if the validation has already been performed on the client. This helps prevent users from being able to bypass validation by impersonating another user or a preapproved transaction.

Client-side validation is enabled by default. If the client is capable, uplevel validation will be performed automatically. To disable client-side validation you may set "remoteValidation" property to false.

6.4. JGroupValidator

JGroupValidator group other sub-validator components to perform multiple validations in one shot. It polls each sub-validator and aggregates the text messages exposed by each one. To obtain each validator message, JGroupValidator will inpect the "groupMessage" property of each validator.

6.5. Displaying Validation Errors

When doValidation() method is called the validator tests the user's input and returns whether is or not valid. If a validation component is in error, an error message may be displayed in the page by that validation component or in a JGroupValitor component elsewhere on the page. The JGroupValidator component is displayed when some of its validators is false. The following example illustrates displaying errors with a JGroupValidator.

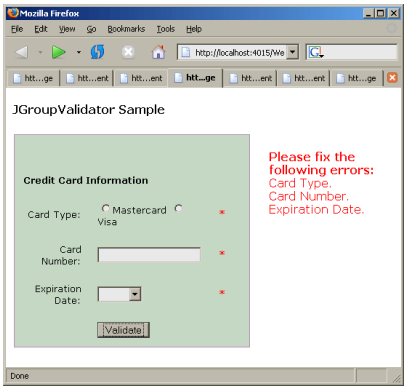

Figure 6.2. JGroupValidatorExample screenshot

```
package net.ar.webonswing.tutorial;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import net.ar.webonswing.*;
import net.ar.webonswing.swing.components.validators.*;
```

```
public class JGroupValidatorExample extends JDialog
{
  public JGroupValidatorExample()
  {
    final JLabel label= new JLabel();

    JRadioButton card1= new JRadioButton("Mastercard");
    JRadioButton card2= new JRadioButton("Visa");
    ButtonGroup cardsGroup= new ButtonGroup();
    cardsGroup.add(card1);
    cardsGroup.add(card2);

    JTextField cardNumberField= new JTextField();

    JComboBox expirationCombo= new JComboBox(new String[]{"", "06/04", "07/04", "08/04", "09/04", "10/04", "11/04",
      "12/04", "01/05", "02/05", "03/05", "04/05", "05/05", "06/05", "07/05", "08/05", "09/05", "10/05", "11/05"
    });

    final JGroupValidator groupValidator= new JGroupValidator();

    final JValidator cardsRequiredFieldValidator= new JRequiredFieldValidator(card1, "*", "Card Type.", true);

    final JValidator expirationRequiredFieldValidator= new JRequiredFieldValidator(expirationCombo, "*",
      "Card Number. ", true);

    final JValidator numberRequiredFieldValidator= new JRequiredFieldValidator(cardNumberField, "*",
      "Expiration Date. ", true);

    groupValidator.addValidator(cardsRequiredFieldValidator);
    groupValidator.addValidator(expirationRequiredFieldValidator);
    groupValidator.addValidator(numberRequiredFieldValidator);

    JButton button= new JButton("Validate");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        if (groupValidator.doValidation())
          label.setText("Page is Valid!");
        else
          label.setText("Some of the required fields are empty");
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));

    getContentPane().add(label).setName("label1");
    getContentPane().add(card1).setName("card1");
    getContentPane().add(card2).setName("card2");
    getContentPane().add(cardsRequiredFieldValidator).setName("cardsValidator");
    getContentPane().add(cardNumberField).setName("number");
    getContentPane().add(numberRequiredFieldValidator).setName("numberValidator");
    getContentPane().add(expirationCombo).setName("expirationCombo");
    getContentPane().add(expirationRequiredFieldValidator).setName("expirationValidator");
    getContentPane().add(button).setName("button");
    getContentPane().add(groupValidator).setName("groupValidator");

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor("JGroupValidatorExample.main"));
  }
}
```

JGroupValidatorExample.html

```
<html>
<head>
</head>

<body>
<span name="main">
<h4><font face="Verdana">JGroupValidator Sample</font></h4>
<p>
<table>
    <tr>
      <td>
      <table bgcolor="#C4D8C4" cellpadding=10 style="border: 1px solid #B78BB7">
      <tr valign="top">
        <td colspan=3>
          <font color="Red"><span name="label1"/></font>
          <br>
        </td>
      </tr>
      <tr>
        <td colspan=3>
        <font face=Verdana size=2><b>Credit Card Information</b></font>
        </td>
      </tr>
      <tr>
        <td align=right>
          <font face=Verdana size=2>Card Type:</font>
```

```
        </td>
        <td>
          <font face=Verdana size=2>
          <input name="card1" type="radio" />
          <span name="card2"/>
          </font>
        </td>
        <td align=middle rowspan=1>
      <font color="Red"><span name="cardsValidator"/></font>
    </td>
      </tr>
      <tr>
        <td align=right>
          <font face=Verdana size=2>Card Number:</font>

        </td>
        <td>
          <input type="text" name="number" />
        </td>
        <td>
          <font color="Red"><span name="numberValidator"/></font>
        </td>

      </tr>
      <tr>
        <td align=right>
          <font face=Verdana size=2>Expiration Date:</font>
        </td>
        <td>
          <span name="expirationCombo" />
        </td>
        <td>
          <font color="Red"><span name="expirationValidator"/></font>
        </td>
        <td>
      </tr>
      <tr>
        <td></td>
        <td>
          <input name="button" value="Validate" />
        </td>
        <td></td>
      </tr>
      </table>
     </td>
        <td valign=top>
            <table cellpadding=20><tr><td>
            <font face="verdana" color="Red" size="3"><span name="groupValidator" /></font>
            </td></tr></table>
        </td>
    </tr>
</table>

</span>
</body>
</html>
```

6.6. Working with JCompareValidator

The JCompareValidator component compares the value of componentToValidate to the value of componentToCompare or a static value provided in valueToCompare. An operator defines the type of comparison to perform--for example, Equal or Not Equal. JCompareValidator performs the validation by evaluating these properties as an expression, as follows:

(componentToValidate <Operator> componentToCompare)

or

(componentToValidate <Operator> valueToCompare)

If the expression evaluates true, the validation result is valid.

The JCompareValidator component could also be used to do datatype validation.For example, if birth date information has to be collected from a user registration page, JCompareValidator component could be used to

make sure that the date is in a recognized format before it is submitted to the database.

The following sample shows how to use the CompareValidator component.

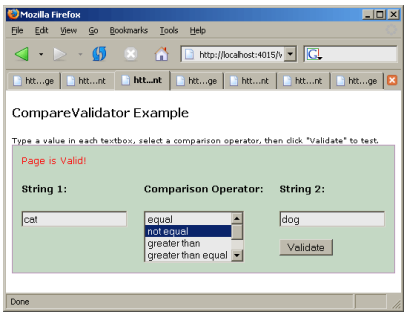

Figure 6.3. JCompareValidatorExample screenshot

```
public class JCompareValidatorExample extends JDialog
{
  public JCompareValidatorExample()
  {
    final JLabel label= new JLabel();

    JTextField component1= new JTextField();
    JTextField component2= new JTextField();

    final JCompareValidator compareValidator= new JCompareValidator(component1, "", "", false,
      component2, JCompareValidator.Operation.equal, JCompareValidator.Type.STRING);

    final JList operationList= new JList(
      new Object[] {
        JCompareValidator.Operation.equal,
        JCompareValidator.Operation.notEqual,
        JCompareValidator.Operation.greaterThan,
        JCompareValidator.Operation.greaterThanEqual,
        JCompareValidator.Operation.lessThan,
        JCompareValidator.Operation.lessThanEqual});

    JButton button= new JButton("Validate");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        compareValidator.setOperation((JCompareValidator.Operation) operationList.getSelectedValue());

        if (compareValidator.doValidation())
          label.setText("Page is Valid!");
        else
          label.setText("Not valid!");
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));

    getContentPane().add(label).setName("label1");
    getContentPane().add(component1).setName("component1");
    getContentPane().add(component2).setName("component2");
    getContentPane().add(operationList).setName("operationList");
    getContentPane().add(compareValidator).setName("compareValidator");
    getContentPane().add(button).setName("button");

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor("JCompareValidatorExample.main"));
  }
}
```

JCompareValidatorExample.html

```
<html>
<head>
</head>

<body>
<span name="main">
    <h4><font face="Verdana">CompareValidator Example</font></h4>
    <font face="Verdana" size="1px">
      Type a value in each textbox, select a comparison operator, then click "Validate" to test.</font>
```

```
      <table bgcolor="#C4D8C4" cellpadding=10 style="border: 1px solid #B78BB7">

      <tr valign="top">
        <td colspan=3>
          <font face="verdana" color="Red" size="2">
            <span name="label1" />
          </font>
        </td>
      </tr>
      <tr valign="top">
        <td>
            <h5><font face="Verdana">String 1:</font></h5>
            <span name="component1" />
        </td>
        <td>
            <h5><font face="Verdana">Comparison Operator:</font></h5>
            <span name="operationList" size="4"/>
        </td>
        <td>
            <h5><font face="Verdana">String 2:</font></h5>
            <span name="component2" /><p>
            <input name="button" />
        </td>
      </tr>
      </table>

      <span name="compareValidator" />
</span>
</body>
</html>
```

6.7. Working with JRangeValidator

The JRangeValidator component tests whether an input value falls within a given range. JRangeValidator uses three key properties to perform its validation. ComponentToValidate contains the value to validate. LowValue and highValue define the minimum and maximum values of the valid range, and also can validate a range delimited by other components using lowComponent and highComponent properties.

This sample shows how to use the JRangeValidator component.

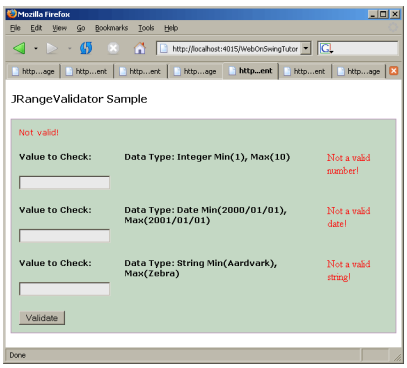

Figure 6.4. JRangeValidatorExample screenshot

```
public class JRangeValidatorExample extends JDialog
{
  public JRangeValidatorExample()
  {
    final JLabel label= new JLabel();

    JTextField component1= new JTextField();
    JTextField component2= new JTextField();
    JTextField component3= new JTextField();

    final JRangeValidator rangeValidator1= new JRangeValidator(component1, "Not a valid number!", "", false, "1",
      "10", JCompareValidator.Type.INTEGER);
    final JRangeValidator rangeValidator2= new JRangeValidator(component2, "Not a valid date!", "", false,
      "2000/01/01", "2001/01/01", JCompareValidator.Type.DATE("yyyy/MM/dd"));
```

```
    final JRangeValidator rangeValidator3= new JRangeValidator(component3, "Not a valid string!", "", false,
      "Aardvark", "Zebra", JCompareValidator.Type.STRING);

    JButton button= new JButton("Validate");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        if (rangeValidator1.doValidation() & rangeValidator2.doValidation() & rangeValidator3.doValidation())
          label.setText("Page is Valid!");
        else
          label.setText("Not valid!");
      }
    });


    getContentPane().setLayout(new GridLayout(0, 1));

    getContentPane().add(component1).setName("component1");
    getContentPane().add(component2).setName("component2");
    getContentPane().add(component3).setName("component3");
    getContentPane().add(rangeValidator1).setName("validator1");
    getContentPane().add(rangeValidator2).setName("validator2");
    getContentPane().add(rangeValidator3).setName("validator3");
    getContentPane().add(button).setName("button");
    getContentPane().add(label).setName("outputLabel");

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor("JRangeValidatorExample.main"));
  }
}
```

JRangeValidatorExample.html

```
<html>
<head>
</head>

<body>
<span name="main">
<h4><font face="Verdana">JRangeValidator Sample</font></h4>
    <p>
      <table bgcolor="#C4D8C4" cellpadding=10 style="border: 1px solid #B78BB7">
      <tr valign="top">
        <td colspan=3>
          <font face="verdana" color="Red" size="2">
            <span name="outputLabel" />
          </font>
        </td>
      </tr>
      <tr valign="top">
        <td>
            <h5><font face="Verdana">Value to Check:</font></h5>
            <span name="component1" />
        </td>
        <td>
            <h5><font face="Verdana">Data Type: Integer Min(1), Max(10)</font></h5>
        </td>
        <td>
             <font color="Red"><span name="validator1" /></font>
        </td>
      </tr>
      <tr valign="top">
        <td>
            <h5><font face="Verdana">Value to Check:</font></h5>
            <span name="component2" />
        </td>
        <td>
            <h5><font face="Verdana">Data Type: Date Min(2000/01/01), Max(2001/01/01)</font></h5>
        </td>
        <td>
             <font color="Red"><span name="validator2" /></font>
        </td>
      </tr>
      <tr valign="top">
        <td>
            <h5><font face="Verdana">Value to Check:</font></h5>
            <span name="component3" />
        </td>
        <td>
            <h5><font face="Verdana">Data Type: String Min(Aardvark), Max(Zebra)</font></h5>
        </td>
        <td>
             <font color="Red"><span name="validator3" /></font>
        </td>
      </tr>
      <tr valign="top">
        <td colspan="3">
         <span name="button" />
```

```
        </td>
      </tr>
     </table>
</span>
</body>
</html>
```

6.8. Working with JRegularExpressionValidator

The JRegularExpressionValidator component checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.

JRegularExpressionValidator uses two key properties to perform its validation. ComponentToValidate contains the value to validate. Re property contains the regular expression to match.

These samples illustrates using the JRegularExpressionValidator component.

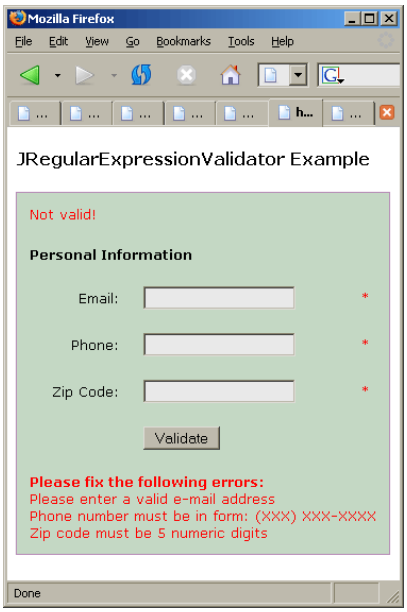

Figure 6.5. JRegularExpressionValidatorExample screenshot

```
public class JRegularExpressionValidatorExample extends JDialog
{
  public JRegularExpressionValidatorExample()
  {
    final JLabel label= new JLabel();

    JTextField component1= new JTextField();
    JTextField component2= new JTextField();
    JTextField component3= new JTextField();

    JValidator requiredFieldValidator1= new JRequiredFieldValidator(component1);
    JValidator requiredFieldValidator2= new JRequiredFieldValidator(component2);
    JValidator requiredFieldValidator3= new JRequiredFieldValidator(component3);

    JValidator regularExpressionValidator1= new JRegularExpressionValidator(component1, "",
      "Please enter a valid e-mail address", false, "^[\\w-]+@[\\w-]+\\.(com|net|org|edu|mil)$");
    JValidator regularExpressionValidator2= new JRegularExpressionValidator(component2, "",
      "Phone number must be in form: (XXX) XXX-XXXX", false,
      "(^x\\s*[0-9]{5}$)|(^(\\([1-9][0-9]{2}\\)\\s)?[1-9][0-9]{2}-[0-9]{4}(\\sx\\s*[0-9]{5})?$)");
    JValidator regularExpressionValidator3= new JRegularExpressionValidator(component3, "",
      "Zip code must be 5 numeric digits", false, "^\\d{5}$");

    final JGroupValidator groupValidator= new JGroupValidator(
      new JValidator[]{requiredFieldValidator1, requiredFieldValidator2, requiredFieldValidator3,
        regularExpressionValidator1, regularExpressionValidator2, regularExpressionValidator3}, false);
```

```
    JButton button= new JButton("Validate");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        if (groupValidator.doValidation())
          label.setText("Page is Valid!");
        else
          label.setText("Not valid!");
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));

    getContentPane().add(component1).setName("component1");
    getContentPane().add(component2).setName("component2");
    getContentPane().add(component3).setName("component3");
    getContentPane().add(requiredFieldValidator1).setName("requiredFieldValidator1");
    getContentPane().add(requiredFieldValidator2).setName("requiredFieldValidator2");
    getContentPane().add(requiredFieldValidator3).setName("requiredFieldValidator3");
    getContentPane().add(regularExpressionValidator1).setName("regularExpressionValidator1");
    getContentPane().add(regularExpressionValidator2).setName("regularExpressionValidator2");
    getContentPane().add(regularExpressionValidator3).setName("regularExpressionValidator3");
    getContentPane().add(button).setName("button");
    getContentPane().add(groupValidator).setName("groupValidator");
    getContentPane().add(label).setName("label");

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor(
      "JRegularExpressionValidatorExample.main"));
  }
}
```

JRegularExpressionValidatorExample.html

```
<html>
<head>
</head>

<body>
<span name="main">
<h4><font face="Verdana">JRegularExpressionValidator Example</font></h4>
<p>

 <table bgcolor="#C4D8C4" cellpadding=10 style="border: 1px solid #B78BB7">
    <tr valign="top">
      <td colspan=3>
        <font face="verdana" color="Red" size="2">
          <span name="label" />
        </font>
      </td>
    </tr>

    <tr>
      <td colspan=3>
      <font face=Verdana size=2><b>Personal Information</b></font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>Email:</font>
      </td>
      <td>
        <span name="component1" />
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="requiredFieldValidator1" />
        <span name="regularExpressionValidator1" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>Phone:</font>
      </td>
      <td>
        <span name="component2" />
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="requiredFieldValidator2" />
        <span name="regularExpressionValidator2" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
```

```
        <font face=Verdana size=2>Zip Code:</font>
      </td>
      <td>
        <span name="component3" />
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="requiredFieldValidator3" />
        <span name="regularExpressionValidator3" />
        </font>
      </td>
    </tr>
    <tr>
      <td></td>
      <td>
        <span name="button" />
      </td>
      <td></td>
    </tr>
    <tr valign="top">
      <td colspan=3>
        <font face="verdana" color="Red" size="2">
          <span name="groupValidator" />
        </font>
      </td>
    </tr>
    </table>

</span>
</body>
</html>
```

6.9. Bringing It All Together

This sample shows a typical registration form, using some of the variations of validation components.

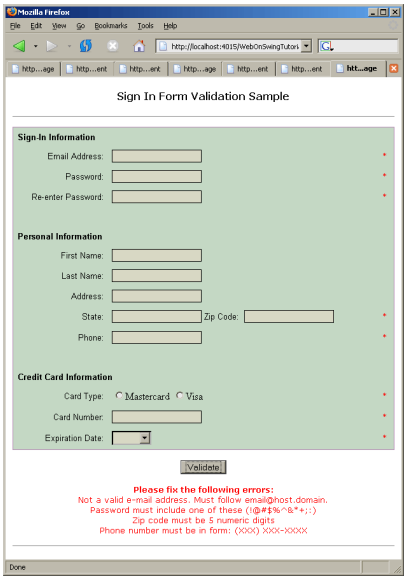

Figure 6.6. ValidatorsExample screenshot

```
public class ValidatorsExample extends JDialog
{
  public ValidatorsExample()
  {
    JComponent email= addNewComponent(new JTextField(), "email");
    JComponent password= addNewComponent(new JPasswordField(), "password");
    JComponent reenterPassword= addNewComponent(new JPasswordField(), "reenterPassword");
    JComponent zip= addNewComponent(new JTextField(), "zip");
    JComponent phone= addNewComponent(new JTextField(), "phone");
    JComponent cardNumber= addNewComponent(new JTextField(), "cardNumber");
    JComponent expirationCombo= addNewComponent(new JComboBox(new String[]{"",
      "06/04", "07/04", "08/04", "09/04", "10/04", "11/04",
      "12/04", "01/05", "02/05", "03/05", "04/05", "05/05",
```

```
      "06/05", "07/05", "08/05", "09/05", "10/05", "11/05", "12/05"}), "expirationCombo");


    addNewComponent(new JTextField(), "firstName");
    addNewComponent(new JTextField(), "lastName");
    addNewComponent(new JTextField(), "address");
    addNewComponent(new JTextField(), "state");


    JRadioButton cardType1= (JRadioButton) addNewComponent(new JRadioButton("Mastercard"), "cardType1");
    JRadioButton cardType2= (JRadioButton) addNewComponent(new JRadioButton("Visa"), "cardType2");
    ButtonGroup cardsGroup= new ButtonGroup();
    cardsGroup.add(cardType1);
    cardsGroup.add(cardType2);


    JValidator emailRequiredValidator= new JRequiredFieldValidator(email);
    JValidator passwordRequiredValidator= new JRequiredFieldValidator(password);
    JValidator reenterPasswordRequiredValidator= new JRequiredFieldValidator(reenterPassword);
    JValidator zipRequiredValidator= new JRequiredFieldValidator(zip);
    JValidator phoneRequiredValidator= new JRequiredFieldValidator(phone);
    JValidator cardsRequiredFieldValidator= new JRequiredFieldValidator(cardType1);
    JValidator expirationRequiredFieldValidator= new JRequiredFieldValidator(expirationCombo);
    JValidator numberRequiredFieldValidator= new JRequiredFieldValidator(cardNumber);

    JValidator emailValidator= new JRegularExpressionValidator(email, "",
      "Not a valid e-mail address.  Must follow email@host.domain.", false,
      "^[\\w-]+@[\\w-]+\\.(com|net|org|edu|mil)$");

    JValidator passwordValidator= new JRegularExpressionValidator(password, "",
      "Password must include one of these (!@#$%^&*+;:)", false, ".*[!@#$%^&*+;:].*");

    JValidator reenterPasswordValidator= new JCompareValidator(reenterPassword, "", "Password fields dont match",
      false, password);

    JValidator zipValidator= new JRegularExpressionValidator(zip, "", "Zip code must be 5 numeric digits", false,
      "^\\d{5}$");

    JValidator phoneValidator= new JRegularExpressionValidator(phone,"","Phone number must be in form: (XXX) XXX-XXXX",
      false, "(^x\\s*[0-9]{5}$)|(^(\\([1-9][0-9]{2}\\)\\s)?[1-9][0-9]{2}-[0-9]{4}(\\sx\\s*[0-9]{5})?$)");

    final JGroupValidator groupValidator= new JGroupValidator(new JValidator[]{ emailRequiredValidator,
      passwordRequiredValidator, reenterPasswordRequiredValidator, zipRequiredValidator, phoneRequiredValidator,
      cardsRequiredFieldValidator, expirationRequiredFieldValidator, emailValidator, numberRequiredFieldValidator,
      passwordValidator, reenterPasswordValidator, zipValidator, phoneValidator}, true);

    addAllValidators(groupValidator);

    final JLabel label= new JLabel();

    JButton button= new JButton("Validate");
    button.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        if (groupValidator.doValidation())
          label.setText("Page is Valid!");
        else
          label.setText("Not valid!");
      }
    });

    getContentPane().setLayout(new GridLayout(0, 1));

    getContentPane().add(groupValidator).setName("groupValidator");
    getContentPane().add(button).setName("button");
    getContentPane().add(label).setName("outputLabel");

    getContentPane().setLayout(WosFramework.getPropagateTemplateLayoutByNameFor("ValidatorsExample.main"));
  }

  protected JComponent addNewComponent(JComponent aComponent, String aName)
  {
    getContentPane().add(aComponent).setName(aName);
    return aComponent;
  }

  protected void addAllValidators(JGroupValidator aGroupValidator)
  {
    for (Iterator i= aGroupValidator.getValidators().iterator(); i.hasNext();)
    {
      JValidator validator= (JValidator) i.next();
      getContentPane().add(validator).setName(validator.getComponentToValidate().getName() +
        WosHelper.getNoPackageClassName(validator));
    }
  }
}
```

ValidatorsExample.html

```
<html>
<head>
</head>

<body>
<span name="main">
    <center>
<h4><font face="Verdana">Sign In Form Validation Sample</font></h4>
    <hr width=600 size=1 noshade>
        <font face="verdana" color="Red" size="2">
          <span name="outputLabel" />
        </font>
    <p>
    <table border=0 width=600 bgcolor="#C4D8C4" cellpadding=5 style="border: 1px solid #B78BB7">
    <tr><td colspan=3>
        <table border=0 cellpadding=0 cellspacing=0 width="100%">
        <tr><td>
            <font face=geneva,arial size=-1><b>Sign-In Information</b></font>
        </td></tr>
        </table>
    </td></tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Email Address:</font>
      </td>
      <td>
        <input name="email" type="text" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="emailJRequiredFieldValidator" />
        <span name="emailJRegularExpressionValidator" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Password:</font>
      </td>
      <td>
        <input name="password" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="passwordJRequiredFieldValidator" />
        <span name="passwordJRegularExpressionValidator" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Re-enter Password:</font>
      </td>
      <td>
        <input name="reenterPassword" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="reenterPasswordJRequiredFieldValidator" />
        <span name="reenterPasswordJCompareValidator" />
        </font>
      </td>
    </tr>
    <tr><td colspan=3> </td></tr>
    <tr><td colspan=3>
        <table border=0 cellpadding=0 cellspacing=0 width="100%">
        <tr><td><font face=geneva,arial size=-1>
            <b>Personal Information</b></font>
        </td></tr>
        </table>
    </td></tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>First Name:</font>
      </td>
      <td>
        <span name="firstName" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Last Name:</font>
      </td>
      <td>
        <span name="lastName" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
      </td>
    </tr>
```

```
    <tr>
      <td align=right>
        <font face=Arial size=2>Address:</font>
      </td>
      <td>
        <span name="address" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>State:</font>
      </td>
      <td>
        <span name="state" style="border: solid 1px; background: #D8D8C4;"/>
        <font face=Arial size=2>Zip Code:</font> 
        <span name="zip" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="zipJRequiredFieldValidator" />
        <span name="zipJRegularExpressionValidator" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Phone:</font>
      </td>
      <td>
        <span name="phone" style="border: solid 1px; background: #D8D8C4;" maxlength="20" />
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="phoneJRequiredFieldValidator" />
        <span name="phoneJRegularExpressionValidator" />
        </font>
      </td>
    </tr>
    <tr><td colspan=3> </td></tr>
    <tr>
      <td colspan=3>
      <font face=Arial size=2><b>Credit Card Information</b></font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Card Type:</font>
      </td>
      <td>
        <span name="cardType1" style="border: solid 1px; background: #D8D8C4;"/>
        <span name="cardType2" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="cardType1JRequiredFieldValidator" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Card Number:</font>
      </td>
      <td>
        <span name="cardNumber" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="cardNumberJRequiredFieldValidator" />
        </font>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>Expiration Date:</font>
      </td>
      <td>
        <span name="expirationCombo" style="border: solid 1px; background: #D8D8C4;"/>
      </td>
      <td>
        <font face="verdana" color="Red" size="2">
        <span name="expirationComboJRequiredFieldValidator" />
        </font>
      </td>
    </tr>
    </table>
    <p>
    <input name="button" />
    <p>
        <font face="verdana" color="Red" size="2">
          <span name="groupValidator" />
```

```
        </font>
    <hr width=600 size=1 noshade>
</span>
</body>
</html>
```

6.10. Summary

- Validation components can be used to validate inputs on any page.
- More than one component can be used on a given input field.
- Client-side validation may be used in addition to server validation to improve form usability.
- The JGroupValidator component can be used to provide centralized error feedback by querying all validation components for error messages
- Simple validation can be performed using the JCompareValidator and JRangeValidator classes. These are commonly used on numeric data.
- Complex pattern validation can be performed using the JRegularExpressionValidator. Pattern validation is useful for strings like names, address, phone numbers, and email addresses.

Chapter 7. Client side listeners, javascript integration

We will often need to optimize the answers of certain client events, trying to do everything that is possible on the client side in javascript instead of travelling with the data to the server. You can develop javascript listeners by implementing the RemoteListener interface on any Swing listener, within this interface you have to provide the name of javascript class and the parametres that are to be passed in javascript listener. They may be of string type and can also pass other components as parameters, referring swing components in javascript by the name assigned to each one of them.

Finally, in javascript, we will extend the listener type that is needed and implement the desired function, interacting with the components and strings passed through the parameters.

For calculator example we add an association with a **component contributor** that will include a .js file with a class that handles the event in javascript. Also RemoteCalculator will implement RemoteListener interface providing the javascript listener (class) name in **getRemoteName** method, and the parameters passed to its constructor in **getRemoteParameters** method.

```
package net.ar.webonswing.tutorial;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JTextField;

import net.ar.webonswing.WosFramework;
import net.ar.webonswing.remote.RemoteListener;
import net.ar.webonswing.swing.layouts.TemplateLayout;

public class RemoteCalculator extends JDialog implements ActionListener, RemoteListener
{
  protected JLabel resultLabel;
  protected JTextField firstNumerField;
  protected JTextField secondNumerField;
  protected JComboBox operationCombo;

  public RemoteCalculator()
  {
    Container contentPane= getContentPane();

    resultLabel= new JLabel("result");
    firstNumerField= new JTextField("");
    secondNumerField= new JTextField("");
    operationCombo= new JComboBox(new String[] { "+", "-", "*", "/"});
    JButton processButton= new JButton("=");

    processButton.addActionListener(this);

    contentPane.setLayout(new TemplateLayout(WosFramework.getKeyPositionTemplateForName("CalculatorTemplate")));
    contentPane.add(firstNumerField, "firstNumber");
    contentPane.add(operationCombo, "operation");
    contentPane.add(secondNumerField, "secondNumber");
    contentPane.add(resultLabel, "result");
    contentPane.add(processButton, "button");

    WosFramework.assignContributor(this, CalculatorContributor.class);
  }

  public void actionPerformed(ActionEvent e)
  {
    float firstNumber= Float.parseFloat(firstNumerField.getText());
    float secondNumber= Float.parseFloat(secondNumerField.getText());
    float result= 0;

    switch (operationCombo.getSelectedItem().toString().charAt(0))
    {
      case '+':
        result= firstNumber + secondNumber;
        break;
      case '-':
        result= firstNumber - secondNumber;
```

```
        break;
      case '*':
        result= firstNumber * secondNumber;
        break;
      case '/':
        result= firstNumber / secondNumber;
        break;
    }

    resultLabel.setText(Float.toString(result));
  }

  public String getRemoteName()
  {
    return "CalculatorActionListener";
  }

  public Object[] getRemoteParameters()
  {
    return new Object[]{resultLabel, firstNumerField, secondNumerField, operationCombo};
  }
}
```

```
package net.ar.webonswing.tutorial;

import net.ar.webonswing.WosFramework;
import net.ar.webonswing.managers.script.ScriptContributionContainer;
import net.ar.webonswing.ui.RootPaneUIContributor;

public class CalculatorContributor extends RootPaneUIContributor
{
  public void doScriptContribution(ScriptContributionContainer aContributionManager)
  {
    aContributionManager.addInclude(WosFramework.getInstance().getCompleteResourcePath() +
      "/js/CalculatorActionListener.js");
  }
}
```

{AppContext}/net/ar/webonswing/resources/CalculatorActionListener.js

```
function CalculatorActionListener (result, firstNumber, secondNumber, operation)
{
  this.inheritFrom= ActionListener;
  this.inheritFrom();

  this.result= result;
  this.firstNumber= firstNumber;
  this.secondNumber= secondNumber;
  this.operation= operation;

  this.actionPerformed= function (anActionEvent)
  {
    var op= getElement(this.operation);
    getElement(this.result).innerHTML= eval (
      getElement(this.firstNumber).value + op.options[op.selectedIndex].text + getElement(this.secondNumber).value);
  };
}
```

Chapter 8. Page state persistence

In typical desktop applications you don't have to worried about the application state, because the active window instance is always alive. But web applications needs to be stateless for reaching a high scalability level. This means that state is not maintained between client requests by default, so the page state must be reconstructed with some mechanism. WebOnSwing Application Framework provides both aproachs for web applications, statefull windows and stateless windows combined with a persistence engine.

8.1. Statefull mode

Statefull mode keeps every window instance in web session and assigns an unique ID to each one, reaching a desktop like behavior in a web environment. And helped by the automatic refreshing of components WebOnSwing can map an entire desktop application to web

8.2. Stateless mode

Stateless mode create a new instance of the window on each request, and use the Persistence Engine to store and restore state in the web page. This mechanism allows to simulate the behavior of a statefull application in a stateless environment (web).

8.2.1. Persistence Contributor

For interacting with Persistence Engine you have to implement **PersistenceContributor** interface in the contributor class associated to the involved component. This interface has methods that will be called by the Persistence Engine to perform the storing and restoring of component state and one method to perform a comparison between current state and stored state. When this comparison said that both values are not equal, if component refreshing is active, current state will be send to the client to update the component view.

8.2.1.1. StatePersistenceExample

The following example ilustrates how PersistenceContributor works.

When the window is constructed, the label's text property will be set to "Hello, World!". Since this property will be set each and every page visit during the window construction, there's no need to persist this information with Persistence Engine. What needs to be persisted is any programmatic changes to the page's state. For example, suppose you have a label component and two buttons, a Change Message Button and a Dummy Button. The Change Message Button has an ActionListener that assigns the label's text property to "Goodbye, Everyone!"; the Dummy Button just causes a server event that doesn't execute any code. The change to the label's text property in the Change Message Button would need to be saved within the page state with Persistence Engine. So LabelPersistenceContributor will store this information in web page, throught PersistenceContributionContainer, and restore this state later.

```
public class StatePersistenceExample extends JDialog
{
  public StatePersistenceExample()
  {
    final JLabel label= new JLabel("Hello, World!");

    JButton changeButton= new JButton("Change Message");
    changeButton.addActionListener(new ActionListener()
    {
```

```
      public void actionPerformed(ActionEvent aE)
      {
        label.setText("Goodbye, Everyone!");
      }
    });

    JButton dummyButton= new JButton("Dummy Button");
    dummyButton.addActionListener(new ActionListener() {public void actionPerformed(ActionEvent aE) {}});

    getContentPane().setLayout(new GridLayout(0, 1));
    getContentPane().add(label);
    getContentPane().add(changeButton);
    getContentPane().add(dummyButton);

    WosFramework.assignContributor(label, LabelPersistenceContributor.class);
  }
  }
```

```
public class LabelPersistenceContributor extends LabelUIContributor
{
  public void doPersistenceContribution(PersistenceContributionContainer aPersistenceManager)
  {
    JLabel label= (JLabel) getJComponent();
    aPersistenceManager.persistValue(theComponent, label.getText());
  }

  public void restorePersistedValue(PersistenceContributionContainer aPersistenceManager)
  {
    JLabel label= (JLabel) getJComponent();
    label.setText((String) aPersistenceManager.restoreValue(theComponent));
  }
}
```

Chapter 9. Contributors

WebOnSwing use the concept of contributors to provide a clean separation between visual components and the final ui target. These contributors will make up the layer that join visual components (such as Swing) and the target GUI (such as HTML), providing the necessary behavior and data to peform the adaptation between both worlds. Each component has an associated contributor, and there are many contributors types such as, rendering contributors, stylesheets and javascript contributors (as includes or code), persistence contributors.

Contributors are asked for its contributions in the differents steps of page/window construction. When it happens the contributors must put its corresponding contribution in a cotribution container passed as an argument.

So, for example, if we are talking about an UI Contributor, the "doRenderingContribution" method will be called and a "RenderingContributionContainer" will be passed; then the contributor will inspect its associated component to extract all required information to create an HTML representation and finally will call the "doContribution" method of contribution container to add the component rendering.

There are many ways to assign contributors to components. You could add an entry to contributors-manager.config.xml, that will create an association to the desire contributor to all component instances of the specified type. And if you want to set an association for a specific instance, you could use "WosFramework.assignContributor" passing component/contributor pair as parameters.

9.1. ComponentUIContributor

This kind of contributors provides a rendering of its associated component for a specific target and handles also the event dispatching related to the component. To make a rendering contribution, UI contributors, must provide a "Visitable" content (such as a Tag, Template or TextContent instance) that will act as the whole rendering.

This contributors may provide also a Tag instance that will be merge to the template tag that acts as the placeholder. And finally must provide an array of initialization data (javascript when we are working with HTML). All this data provided to create the whole component rendering may create some elements that could fire events. This "client side" events will get back to this same contributor throught the "dispatchEvents" method, that recieves an array of "RemoteEvent" objects. WebOnSwing will group all remote events that start with the uniqueName of each component as its prefix. So you must set this prefix to all "request" parameters (input html tags) that want to be catched by this contributor.

```
public class ButtonUIContributor extends AbstractSwingComponentUIContributor
{
  public void doRenderingContribution(RenderingContributionContainer theContribManager)
  {
    Button aButton= (Button) theComponent;

    Tag tag= new Tag("input");
    tag.setNeedsClosure(false);
    tag.addAttribute(new TagAttribute("type", "button"));
    tag.addAttribute(new TagAttribute("name", theComponent.getUniqueName()));
    tag.addAttribute(new TagAttribute("value", aButton.getText()));
    tag.addAttribute(new TagAttribute("onclick", "ed.dispatch(new ActionEvent(this.name, ''));"));

    Template theTemplate= GuiaHelper.getTemplateForComponent("JButton", aButton);
    theTemplate.addElement(new IdTagTemplateElement("theButton", theTag, theTag));

    theContribManager.doContribution(theComponent, theTemplate, theTag,
      new String[]{"getComponent ('"+theComponent.getUniqueName()+"').addListener(new ActionListener());"});
  }

  public void dispatchEvents(List anEvents)
  {
    for (Iterator i= anEvents.iterator(); i.hasNext();)
    {
      RemoteEvent theEvent= (RemoteEvent) i.next();
```

```
      Button button= (Button) theEvent.getSource();

      if (theEvent.getType().equals("actionPerformed"))
        button.doClick();
    }
  }
}
```

9.2. StyleContributor and ScriptContributor

Both types of contributors could add data as an including file (.css or .js) and also could add single code parts, such as stylesheets classes that will be put inside a "style" tag in html "head", or javascript code that will be added before html body surrounded by a script tag.

```
public void doScriptContribution(ScriptContributionContainer aContributionManager)
{
  aContributionManager.addInclude("/js/validators/JValidator.js");
  aContributionManager.addCode("var clientValidation= true;");
}
```

9.3. PersistenceContributor

The persistence contributor has the mision of providing to WebOnSwing the data that the associated component want to preserve between sucesive requests. This data will be extracted from the component and calling "doPersistenceContribution" method of persistence container will be stored to be used in page generation.

When a request is recieved in future interactions, this preserve data will be available in this container, and this contributor will be able to get this throught "restorePersistedValue" method.

There is also an additional method in this interface that help the framework to know if already persisted data is equal or not to the current one. This information is used by the component refreshing mechanism (partial updates) to realize which component must be refreshed.

```
public class RefreshProgressBarUIContributor extends ProgressBarUIContributor
{
  public void doPersistenceContribution(PersistenceContributionContainer aPersistenceManager)
  {
    JProgressBar aProgressBar= (JProgressBar)theComponent;
    aPersistenceManager.persistValue(theComponent, new Integer(aProgressBar.getValue()));
  }

  public boolean isPersistedValueEqualToModel(PersistenceContributionContainer aPersistenceManager)
  {
    JProgressBar aProgressBar= (JProgressBar)theComponent;
    Integer value= (Integer)aPersistenceManager.restoreValue(theComponent);
    return value.intValue() == aProgressBar.getValue();
  }

  public void restorePersistedValue(PersistenceContributionContainer aPersistenceManager)
  {
    Integer value= (Integer)aPersistenceManager.restoreValue(theComponent);
    JProgressBar aProgressBar= (JProgressBar)theComponent;
    aProgressBar.setValue(value.intValue());
  }
}
```

Chapter 10. Cookies and web session management

not documented yet, see weblog app

Chapter 11. HtmlPage behaviour

Not documented yet in english

see Comportamiento de HtmlPage [http://webonswing.sourceforge.net/index/news-app/story.2]

Chapter 12. Single component refresh or partial updates of html page

12.1. Simulating desktop applications by periodic pollings

not documented yet

Not documented yet

Chapter 13. Wrapping others component based application frameworks

13.1. Interfaces that you have to implement

Not documented yet, see Swing wrapping classes

13.2. Wrapping Swing

Available but not documented yet, see Swing wrapping classes

13.2.1. HTML/Swing components mapping and equivalences

13.3. Wrapping SWT

Partialy available but not documented yet, see SWT wrapping examples in WebOnSwingDemo

13.4. Wrapping Web Forms

Not available yet

13.5. Wrapping SWF

Not available yet

Chapter 14. Constructing your own component hierarchy

14.1. Your own visual components

14.2. Your own WindowManager

14.3. Your own ComponentNameManager