JavaServer Faces - Converter

This tutorial explains the usage of converters in JSF. It shows how you can use standard converters step by step using an example application.

General

Author:
Sascha Wolski
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
March, 10 2005

Software:
Eclipse 3.x
MyEclipse 3.8.x

Source code:
The sources do not contain any project files of eclipse or libraries. Create a new project following the tutorial, add the libraries as explained in the tutorial and then you can copy the sources in your new project.
http://www.laliluna.de/assets/tutorials/jsf-converter-source-light.zip

PDF Version
http://www.laliluna.de/assets/tutorials/jsf-converter-en.pdf

What is a converter

A converter is used to format an object to a “nicer” text to be displayed. For example if you want to display a date in JSP you can use a converter to format the date in terms the user understand, like “10/03/2005”.

But there is another way to use a converter. If you use them in conjunction with an input control, the user's input must be in the format specified by the converter. If the format of the input is not matching the format you can throw an exception in the converter which is displayed to the user. The associated object is not updated.

Using converters

Converter can be specified within JSP, or you can register converters programmatically.

You can register a converter using JSP in one of the three ways:

Specify the converter identifier with a component tag's *converter* property.

```
<h:outputText value="#{myBean.date}" converter="myConverter">
```

Nest the <f:converter> tag the converters's identifier inside the component tag.

```

<h:outputText value="#{myBean.date}">
    <f:converter converterId="myConverter"/>
</h:outputText>
```

Nest the converter's custom tag inside a component tag.

```
<h:outputText value="#{myBean.date}">
    <laliluna:myConverter />
</h:outputText>
```

The following JSF Tags supports converters.

```
<h:outputText>
```

```
<h:outputFormat>
<h:outputLink>
<h:outputLabel>
<h:inputText>
<h:inputTextarea>
<h:inputHidden>
<h:inputSecret>
<h:selectBooleanCheckbox>
<h:selectManyListbox>
<h:selectMaynyMenu>
<h:selectOneRadio>
<h:selectOneListbox>
<h:selectOneMenu>
```

If you don't specify a converter, JSF will pick one for you. The framework has standard converters for all of the basic types: *Long*, *Byte*, *Integer*, *Short*, Character, *Double*, *Float*, *BigDecimal*, *BigInteger* and *Boolean*.. For example, if your component is associated with a property of type *boolean*, JSF will choose the *Boolean* converter. Primitive types are automatically converted to their object counterparts.

Using standard converters

DateTime converter

For all basic Java types JSF will automatically use converters. But if you want to format a *Date* object JSF provides a converter tag *<f:convertDateTime>*. This tag must be nested inside a component tag that supports converters.

```
<h:outputText value="#{myBean.date}">
      <f:convertDateTime type="date" dateStyle="medium"/>
</h:outputText>
```

The converter *DateTime* supports attributes, like *type* or *dateStyle*, to configure the converter. The listing below shows the attributes you can use with the *DateTime* converter.

Attribute name	***Description***
dateStyle	Specifies the formatting style for the date portion of the string. Valid options are *short*, *medium* (default), *long* and *full*. Only valid if attribute *type* is set.
timeStyle	Specifies the formatting style for the time portion of the string. Valid options are *short*, *medium* (default), *long* and *full*. Only valid if attribute *type* is set.
timeZone	Specifed the time zone for the date. If not set, Greenwich mean time (GMT) will be used.
locale	The specifed local to use for displaying this date. Overrides the user's current locale
pattern	The date format pattern used to convert this number. Use this or the property *type*.
type	Specifies whether to display the date, time or both.

Number converter

The second converter you can customize by using additional attributes is the *Number* converter. It is useful for displaying numbers in basic formats that works for the user's locale.

```
<h:outputText value="#{myBean.date}">
      <f:convertNumber type="number" maxIntegerDigits="3"/>
</h:outputText>
```

The listing below shows the attribute you can use with *Number* converter. These attributes allow for precise control how a number is displayed.

Attribute name	***Description***
currencyCode	Specifies a three-digit international currency code when the attribute *type* is *currency*. Use this or *currencySymbol*.
currencySymbol	Specifies a specific symbol, like “$”, to be used when the attribute type is *currency*. Use this or *currencyCode*.
groupingUsed	True if a grouping symbol, like “,” or “ “ should be used. Default is *true*.
integerOnly	True if only the integer portion of the input value should be processed (all decimals will be ignored). Default is *false*.
locale	The locale to be used for displaying this number. Overrides the user's current locale
minFractionDigits	Minimal numbers of fractional digits to display.
maxFractionDigits	Maximal numbers of fractional digits to display.
minIntegerDigits	Minimal numbers of integer digits to display.
maxIntegerDigits	Maximal numbers of integer digits to display.
pattern	The decimal format pattern used to convert this number. Use this or attribute *type*.
type	The type of number; either number (default), currency, or percent. Use this or the attribute *pattern*.

The example Application

Create a new java project named *JSFConverter* and add the JSF capabilities.

Backing Bean

Create a new Java class named *MyBean* in the package *de.laliluna.tutorial.converter.bean*.

Add three properties, *dateVar* of type Date, *integerVar* of type Integer and *floatVar* of type Float.

Initial the properties with some values.

Provide getter and setter methods for each properties.

The following source code shows the Java class *MyBean*:

```
public class MyBean {

      private Date dateVar = new Date();
      private Integer integerVar = new Integer(2023);
      private Float floatVar = new Float(9.099783);

      public Date getDateVar() {
            return dateVar;
      }
      public void setDateVar(Date dateVar) {
            this.dateVar = dateVar;
      }
      public Float getFloatVar() {
            return floatVar;
      }
      public void setFloatVar(Float floatVar) {
            this.floatVar = floatVar;
      }
      public Integer getIntegerVar() {
            return integerVar;
      }
      public void setIntegerVar(Integer integerVar) {
            this.integerVar = integerVar;
      }
}
```

Application configuration file

Open the *faces-config.xml* and add a managed bean associated with the backing bean *MyBean*.

Add a navigation-rule for the JSP file *example.jsp*.

The following source code shows the content of the *faces-config.xml*:

```
<faces-config>
      <managed-bean>
            <managed-bean-name>myBean</managed-bean-name>
            <managed-bean-
class>de.laliluna.tutorial.converter.bean.MyBean</managed-bean-class>
            <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>
            <navigation-rule>
                  <from-view-id>example.jsp</from-view-id>
            </navigation-rule>
</faces-config>
```

The JSP file

Create a new JSP file named *example.jsp* in the folder */WebRoot* of the project.

Add the following example usages of converters for the DateTime and Number converter:

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <title>JSF Converters</title>
</head>

<body>
      <f:view>
            <h4>DateTime converter</h4>

            <p>Display only the date and the dateStyle is <i>short</i></p>
            <h:outputText value="#{myBean.dateVar}">
                  <f:convertDateTime type="date" dateStyle="short" />
            </h:outputText>

            <p>Display only the time and the timeStyle is <i>full</i></p>
            <h:outputText value="#{myBean.dateVar}">
                  <f:convertDateTime type="time" timeStyle="full" />
            </h:outputText>

            <p>Display both date and time, the dateStyle is <i>full</i> and the
locale is <i>ru</i></p>
            <h:outputText value="#{myBean.dateVar}">
                  <f:convertDateTime type="both" dateStyle="full" locale="ru" />
            </h:outputText>

            <p>Display the both, date and time and the dateStyle is
<i>short</i></p>
            <h:outputText value="#{myBean.dateVar}">
                  <f:convertDateTime type="both" dateStyle="short" />
            </h:outputText>

            <p>Display a date with the pattern <i>dd.MM.yyyy HH:mm</i></p>
            <h:outputText value="#{myBean.dateVar}">
                  <f:convertDateTime pattern="dd.MM.yyyy HH:mm" />
            </h:outputText>

            <h:form id="datetime1">
                  <p>Input a date and the dateStyle is <i>short</i></p>
                  <h:inputText value="#{myBean.dateVar}">
                        <f:convertDateTime type="date" dateStyle="short" />
                  </h:inputText>
                  <h:commandButton value="Send" />
            </h:form>
            <h:form id="datetime2">
                  <p>Input a date matching this pattern <i>dd.MM.yyyy</i></p>
                  <h:inputText value="#{myBean.dateVar}">
                        <f:convertDateTime pattern="dd.MM.yyyy" />
                  </h:inputText>
                  <h:commandButton value="Send" />
            </h:form>

            <h4>Number converter</h4>

            <p>Display maximal <i>3 integer digits</i></p>
            <h:outputText value="#{myBean.integerVar}">
                  <f:convertNumber maxIntegerDigits="3" />
            </h:outputText>

            <p>Display type is <i>currency</i> and the currencySymbol is
<i>$</i></p>
            <h:outputText value="#{myBean.integerVar}">
                  <f:convertNumber type="currency" currencySymbol="$"/>
            </h:outputText>

            <p>Display type is <i>percent</i></p>
            <h:outputText value="#{myBean.integerVar}">
                  <f:convertNumber type="percent"/>
            </h:outputText>

            <p>Display maximal 4 fraction digits</p>
            <h:outputText value="#{myBean.floatVar}">
                  <f:convertNumber maxFractionDigits="4"/>
            </h:outputText>

            <p>Display number matching pattern <i>###0,00%</i></p>
            <h:outputText value="#{myBean.floatVar}">
                  <f:convertNumber pattern="###0,00%"/>
            </h:outputText>

            <h:form id="number1">
                  <p>Input a number but only the integer digits will be
processed</p>
                  <h:inputText value="#{myBean.integerVar}">
                        <f:convertNumber integerOnly="true"/>
                  </h:inputText>
                  <h:commandButton value="Send" />
            </h:form>

            <h:form id="number2">
                  <p>Input a number matching the pattern <i>##0,00</i></p>
                  <h:inputText value="#{myBean.integerVar}">
                        <f:convertNumber pattern="##0,00"/>
                  </h:inputText>
                  <h:commandButton value="Send" />
            </h:form>

      </f:view>
</body>
```

```
</html>
```

Thats all. Now you can deploy and test the example application. We recommends a Jboss and Tomcat installation. Call the example with the following link:

http://localhost:8080/JSFConverters/example.faces