JavaServer Faces – Developing custom converters

This tutorial explains how to develop your own converters. It shows the usage of own custom converter tags and overriding standard converter of basic types.

General

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

Date:
March, 17 2005

Software:
Eclipse 3.x
PDF

http://www.laliluna.de/download/jsf-custom-converters-en.pdf

Sources

http://www.laliluna.de/download/jsf-custom-converter-source.zip

Requirements

This tutorial extends the JavaServer Faces – Converter tutorial and do not explain what are converters and how are the converters so important for qualified web applications.

Converter class

An own converter class is a normal Java class which implements the Converter interface found in the *javax.faces.convert* package. Converters have identifiers that are registered in a JSF configuration file and often exposed through a constant:

```
public final static String CONVERTER_ID = "laliluna.CreaditCard";
```

This constant can be used to create new instances of your converter. The converter interface provides two methods. The *getAsObject(..)* method to convert a String into a Object and the method *getAsString(..)* to convert a Object into a String.

The following listing shows the two methods of the converter interface:

```
public Object getAsObject(FacesContext facesContext,
                         UIComponent uIComponent,
                         String string) throws ConverterException;

public Object getAsString(FacesContext facesContext,
                         UIComponent uIComponent,
                         Object object) throws ConverterException;
```

You have to throw a new *ConverterException* if conversion is not possible. This exception can be displayed in the JSP file with the *<h:messages>* or *<h:message>* element.

Registration of converters

You can register a new converter by using the *<converter>* element within the application configuration file (*faces-config.xml*). You can register a converter by identifier or type. Here's an example of registering it by identifier:

```
<converter>
      <converter-id>laliluna.CreditCard</converter-id>
      <converter-class>
```

```
            myPackage.CreditCardConverter
      </converter-class>
</converter>
```

The next example shows a registration by type:

```
<converter>
      <converter-for-class>java.lang.Integer</converter-for-class>
      <converter-class>
            myPackage.IntegerConverter
      </converter-class>
</converter>
```

If your custom tag supports additional properties you can add them with the <property> element. The following example shows a more complex registration of a converter that supports properties.

```
<converter>
      <converter-id>laliluna.CreditCard</converter-id>
      <converter-class>myPackage.UserConverter</converter-class>
      <property>
            <description>
                  Defines the character displayed
                  within the credit card number
            </description>
            <property-name>displayCharacter</property-name>
            <property-class>java.lang.String</property-class>
            <default-value>-</default-value>
      </property>
</converter>
```

JSP integration

If you register a converter by identifier you can use the *<f:converter>* element to assign a converter to a component. Here's an example how to assign the CreditCardConverter to a component.

```
<h:outputText value="#{myBean.creditCard}">
      <f:converter converterId="laliluna.CreditCard" />
</h:outputText>
```

If you choose type registration the converter will be automatically used for all component which are associated with this type. For example if you add a component which is associated with a integer value the *IntegerConverter* of the registration example above will be used.

The last way to add a converter in JSP is using a custom tag. Of course, in cases where your converter have properties, or you prefer that front-end developers use a specific tag as opposed to a generic tag and identifier, you can nest the custom tag inside the component element:

```
<h:outputText value="#{myBean.creditCard}">
      <laliluna:creditCardConverter displayCharacter="/" />
</h:outputText>
```

ConvertTag

The ConverterTag class is needed to develop your own custom converter tag. You have to create a new Java class which extends the ConverterTag class.

Use the method below to specify the identifier of the converter that is associated with this tag.

```
public void setConverterId(String converterId);
```

Call the *setConverterId(..)* method with the identifier constant of your converter class within the constructor of your ConverterTag class. Here's an example how to set the identifier for the

CreditCardConverter within the constructor:

```
public CreditCardConverterTag(){
      super();
      setConverterId(CreditCardConverter.CONVERTER_ID);
}
```

In the next method below the converter instance will be created.

```
protected Converter createConverter() throws JspException;
```

Here you can set the properties given by the custom converter tag. For example the custom tag has a property *displayCharacter*, to set a character which is displayed within the credit card number, like “1234-12345-123”, add a property with the same name in the ConverterTag class.

Here's an example which shows the *createConverter(..)* method:

```
protected Converter createConverter() throws JspException {

            CreditCardConverter converter = 
(CreditCardConverter)super.createConverter();
            converter.setDisplayCharacter(displayCharacter);

            return converter;
}
```

The last method you have to override is the *release()* method of the ConverterTag class. Inside the method you can reset any instance variables to their default values:

```
public void release() {
      super.release();
      displayCharacter = null;
}
```

Create a custom tag library

To provide the custom tag to front-end developer you have to create a new custom tag library. A custom tag library is a file with the extension *tld* and places in the folder */WebRoot/WEB-INF/* of your project.

The following example shows the *credit_card.tld* file:

```
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
      <tlib-version>1.0</tlib-version>
      <jsp-version>1.2</jsp-version>
      <short-name>laliluna</short-name>
      <uri>credit-card-converter</uri>
      <description>
        Credit card custom tag library
      </description>
      <tag>
            <name>creditCardConverter</name>
            <tag-class>
                  myPackage.CreditCardConverterTag
            </tag-class>
            <body-content>JSP</body-content>
            <attribute>
                  <name>displayCharacter</name>
                  <required>true</required>
                  <rtexprvalue>flase</rtexprvalue>
            </attribute>
      </tag>
</taglib>
```

Inside the *<tag>* element you can specify the name of the custom tag and the full qualifier class name of your ConverterTag class. With the *<attribute>* element you can set attributes supports by the custom tag.

Include the tag library in your JSP file like the following:

```
<%@ taglib uri="credit-card-converter" prefix="laliluna" %>
```

The example application

Now we want to create a little step by step example which helps to understand the explanation above.

Create a new Java project and add the JSF capabilities with MyEclipse or add the libraries manually to the project. You can find the libraries at http://java.sun.com/j2ee/javaserverfaces/index.jsp

Backing Bean

Create a new Java class named *MyBean* which represent the backing bean in the package *de.laliluna.tutorial.customconverter.bean*.

Add two properties, *creditCard* of type Long and *integerVar* of type Integer.

Provide a getter and setter method for each property.

Here's is the content of the class *MyBean*:

```
public class MyBean {

      //properties
      private Long creditCard;
      private Integer integerVar = new Integer(3);

      //getter and setter methods
      public Integer getIntegerVar() {
            return integerVar;
      }
      public void setIntegerVar(Integer integerVar) {
            this.integerVar = integerVar;
      }
      public Long getCreditCard() {
            return creditCard;
      }
      public void setCreditCard(Long creditCard) {
            this.creditCard = creditCard;
      }
}
```

Converter class

Create a new Java class named *CreditCardConverter* which implements the Converter interface in the package *de.laliluna.tutorial.customconverter.converter*.

Add a constant CONVERTER_ID which holds the converter identifier.

Add a property displayCharacter which holds the split character for the credit card and provide a getter and setter method.

Implement the two methods, *getAsObject(..)* and *getAsString(..)* of the Converter interface.

Within the first method *getAsObject(..)* we convert the text giving by user's input into a Long object. The input string have to match a regular expression. If it do not match throw a new ConverterException.

Within the second method *getAsString(..)* we convert the Long object into a string and format the string to display it to the user, like “1234-12345-123”.

The following example shows the class *CreditCardConverter*:

```
public class CreditCardConverter implements Converter{

      public final static String CONVERTER_ID = "laliluna.CreditCard";

      private String displayCharacter;

      public String getDisplayCharacter() {
            return displayCharacter;
      }
      public void setDisplayCharacter(String displayCharacter) {
            this.displayCharacter = displayCharacter;
      }

      /**
       * Convert from String into Object
       */
      public Object getAsObject(FacesContext context,
                                            UIComponent component,
                                            String string) throws
ConverterException {

            Long lValue = null;

            //check the string with regular expression
            Pattern pattern = Pattern.compile("[0-9]{4}" +
                                          displayCharacter + "[0-9]{5}" +
                                          displayCharacter + "[0-9]{3}");
            Matcher matcher = pattern.matcher(string);

            if(matcher.matches()){
                  //replace the characters and convert the String into a Long
                  lValue = Long.valueOf(string.replaceAll(displayCharacter,
""));
            }else{
                  //if string do not match the regular expression
                  //throw a converter exception
                  throw new ConverterException(
                              new FacesMessage("Conversion into object failed,
credit card number incorrect"));
            }

            return lValue;
      }

      /**
       * Convert from Object into String
       */
      public String getAsString(FacesContext context,
                                            UIComponent component,
                                            Object object) throws
ConverterException {

            String sValue = object.toString();
            if(sValue.length() == 12){
                  //format the returned String object
                  sValue = sValue.substring(0, 4) + displayCharacter +
                               sValue.substring(4, 9) + displayCharacter +
                               sValue.substring(9, 12);
            } else {
                  throw new ConverterException(
                              new FacesMessage("Conversion into string failed,
credit card number incorrect"));
            }

            return sValue;
      }
}
```

Create a second Converter class named *IntegerConverter*, which is used to illustrate how you can use different converters for basic types. The Converter class do nothing else as the standard *IntegerConverter* class do. So copy and paste the source code.

The following source code shows the *IntegerConverter* class:

```
public class IntegerConverter implements Converter{

      public final static String CONVERTER_ID = "laliluna.Integer";

      /**
       * Convert from String into Object
       */
      public Object getAsObject(FacesContext context,
                                          UIComponent component,
                                          String string) {

            Integer iValue;
            try{
                  iValue = Integer.valueOf(string);
            } catch(NumberFormatException e){
                  throw new ConverterException(new FacesMessage("Convertion into
integer failed"));
            }

            return iValue;
      }

      /**
       * Convert from Object into String
       */
      public String getAsString(FacesContext context,
                                          UIComponent component,
                                          Object object) {

            String sValue = object.toString();

            return sValue;
      }
}
```

ConverterTag class

To provide a custom tag for the credit card converter, we have to create a class which extends the *ConverterTag*. Create a new Java class CreditCardConverterTag which extends the ConverterTag class in the package *de.laliluna.tutorial.customconverter.converter*.

Add a property *displayCharacter* which holds the split character of the credit card and provide a getter and setter method.

Implement the constructor of the class and set inside the converter the *CONVERTER_ID* of the *CreditCardConverter* with the *setConverterId* method.

Override the *release()* of the *ConverterTag* class and reset the property *displayCharacter* to null.

Override the *createConverter()* method. Within the method create a new *CreditCardConverter* instance and set the property *displayCharacter* of the *CreditCardConverter* class.

The following source code shows the CreditCardConverterTag class:

```
public class CreditCardConverterTag extends ConverterTag {


	private String displayCharacter;

	public String getDisplayCharacter() {
		return displayCharacter;
	}
	public void setDisplayCharacter(String displayCharacter) {
		this.displayCharacter = displayCharacter;
	}

	/**
	 * Constructor
	 */
	public CreditCardConverterTag(){
		super();
		setConverterId(CreditCardConverter.CONVERTER_ID);
	}

	/**
	 * release method
	 */
	public void release() {
		super.release();
		displayCharacter = null;
	}

	/**
	 * createConverter method
	 */
	protected Converter createConverter() throws JspException {

		CreditCardConverter converter =
(CreditCardConverter)super.createConverter();
		converter.setDisplayCharacter(displayCharacter);

		return converter;
	}
}
```

Custom tag library

To provide the custom tag to the front-end developer you have to create new file named credit_card.tld in the folder */WebRoot/WEB-INF* of your project.

With the *<short-name>* element you specify the default prefix of the custom tag (`<laliluna:credit...>`).

The *<uri>* element specify the public URI that identifies the custom tag. You need the URI to include the tag library in your JSP file.
(`<%@ taglib uri="credit-card-converter" prefix="laliluna" %>`).

With the *<name>* element nested inside the *<tag>* element you specify the name of the custom tag. The element *<tag-class>* specify the full qualified class name of your *ConverterTag* class.

With the *<attribute>* element you can add attributes, the front-end developer can use to configure the converter. In your case the developer can change the split character of the credit card number.

```
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
      <tlib-version>1.0</tlib-version>
      <jsp-version>1.2</jsp-version>
      <short-name>laliluna</short-name>
      <uri>credit-card-converter</uri>
      <description>
        Credit card custom tag library
      </description>
      <tag>
            <name>creditCardConverter</name>
            <tag-class>

de.laliluna.tutorial.customconverter.converter.CreditCardConverterTag
            </tag-class>
            <body-content>JSP</body-content>
            <attribute>
                  <name>displayCharacter</name>
                  <required>true</required>
                  <rtexprvalue>false</rtexprvalue>
            </attribute>
      </tag>
</taglib>
```

Application configuration file (faces-config.xml)

Now we want to configure the faces-config.xml. Open the file and first add a navigation rule for the the *example.jsp* file we create later.

Add a managed bean myBean for the backing bean class *MyBean*.

Add a converter registration for the *CreaditCardConverter* class and add the property *displayCharacter* with the element *<porperty>*.

Add a second converter registration, which associated with the basic Java type *Integer*. We want to show how to assign a converter by type. So if you add a component in your JSP which associated with an integer property this converter will be automatically used. Use the *IntegerConverter*.

```
<faces-config>
      <navigation-rule>
            <from-view-id>example.jsp</from-view-id>
      </navigation-rule>

      <managed-bean>
            <managed-bean-name>myBean</managed-bean-name>
            <managed-bean-
class>de.laliluna.tutorial.customconverter.bean.MyBean</managed-bean-class>
            <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>

      <converter>
```

```
            <converter-id>laliluna.CreditCard</converter-id>
            <converter-
class>de.laliluna.tutorial.customconverter.converter.CreditCardConverter</conver
ter-class>
            <property>
                  <description>
                        Defines the character displayed
                        within the credit card number
                  </description>
                  <property-name>displayCharacter</property-name>
                  <property-class>java.lang.String</property-class>
                  <default-value>-</default-value>
            </property>
      </converter>

      <converter>
            <converter-for-class>java.lang.Integer</converter-for-class>
            <converter-
class>de.laliluna.tutorial.customconverter.converter.IntegerConverter</converter
-class>
      </converter>
</faces-config>
```

JSP file

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

The first two components are associated with the property *integerVar* of the backing bean. Here you do not assign any converter because we have a registration by type in the application configuration file.

The second two components are associated with the property *creditCard* of the backing bean. These are the credit card output and input components and we have to assign the credit card converter.

We do this with nesting the custom tag we created for the *CreditCardConverter*. Configure the converter of the output component to display the “-” character as credit card number splitter. For the input of the credit card number the user must use another character, like “/”.

Display the ConverterExceptions by using the *<h:messages>* element. You can also use the *<h:message>* element and specify the *for* attribute.

The following source code shows the content of the JSP file:

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

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

<body>
      <f:view>
            <p>
            <h:outputText value="#{myBean.integerVar}" />
            </p>

            <h:form id="integerVar">
                  <p>
                  <h:inputText value="#{myBean.integerVar}" />
                  </p>
                  <h:commandButton value="Submit"/>
            </h:form>

            <p>
            <h:outputText value="#{myBean.creditCard}">
                  <laliluna:creditCardConverter displayCharacter="-" />
            </h:outputText>
            </p>
            <h:form id="user">
                  <p>
                  <h:inputText value="#{myBean.creditCard}">
                        <laliluna:creditCardConverter displayCharacter="/" />
                  </h:inputText>
                  <i>input format: 1234/12345/123</i>
                  </p>
                  <h:commandButton value="Submit"/>
            </h:form>

            <h:messages />
      </f:view>
</body>
</html>
```

Now that's all, you can deploy and test the little example application. We recommend a Jboss or Tomcat installation. Call the project with the following link:

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