SAP NetWeaver
How-To Guide

How To... Develop Custom Converters and Validators

Applicable Releases:

SAP NetWeaver Composition Environment 7.1

Topic Area:

User Productivity

Development and Composition

Capability:

User Interface Technology

Java

Version 1.0

October 2008

THE BEST-RUN BUSINESSES RUN SAP™

© Copyright 2008 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided “as is” without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

SAP NetWeaver “How-to” Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

Any software coding and/or code lines / strings (“Code”) included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

Disclaimer

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

THE BEST-RUN BUSINESSES RUN SAP™

Document History

Document Version	Description
1.00	First official release of this guide

THE BEST-RUN BUSINESSES RUN SAP™

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
⚠	Caution
💡	Note or Important
✿	Example
⬆	Recommendation or Tip

THE BEST-RUN BUSINESSES RUN SAP™

Table of Contents

1. Business Scenario

In the previous tutorial you created a JSF application that used standard converters and validators to maintain the integrity of your model.

The following guide will extend the Product Offer tutorial (Convert And Validate Data [extern]) by adding a *Product Code* field. It will also show you how to implement application-specific converters and validators. The custom converter will allow users to enter the code without any special characters and it will format the code using a dash to separate the characters. The custom validator will check the digits of the code, so it will have three characters follow by a dash and four numbers (XXX-1234).

If there are any errors, the page is redisplayed with the values entered by the user in order to correct them. If all validations are successful, the application will continue with the creation process.

2. Background Information

JSF standard converters and validators cover a lot of bases, but many web applications must go further. The real power of the Faces conversion model is its extensibility and you will now be familiarized with the creation of custom converter class.

Creating custom Validator implementations is even more common than creating custom Converter instances. JSF makes it very easy to create a custom validator class. This guide will also explain how to create and register a custom validator class.

3. Prerequisites

The following is a list of all you need for developing JSF applications.

- AS Java 7.1 (CE 7.1 or NW 7.1)
- NWDS 7.1 (SP3 or higher with latest patch level).

Note

While this tutorial is geared towards to the SAP AS Java (the build/deploy steps of the guide), it wouldn't be hard to replace the build/deploy portions with similar steps for any other Java EE 5 platform

Knowledge

- You have a basic knowledge of Java Enterprise Edition
- You have acquired some basic experience with JSF applications, for example by working through the JSF tutorials (Create a Hello World Application using JavaServer Faces [Extern] and Create Your First JSF Application [Extern])
- You have successfully completed the Product Offer tutorial (Convert And Validate Data [extern]) or you have imported the Product Offer Project Template into the SAP NetWeaver Developer Studio

4. Step-by-Step Procedure

In the following sections you will extend the Product Offer tutorial (Convert And Validate Data [extern]) by adding a *Product Code* field.

You will also create a custom converter class by

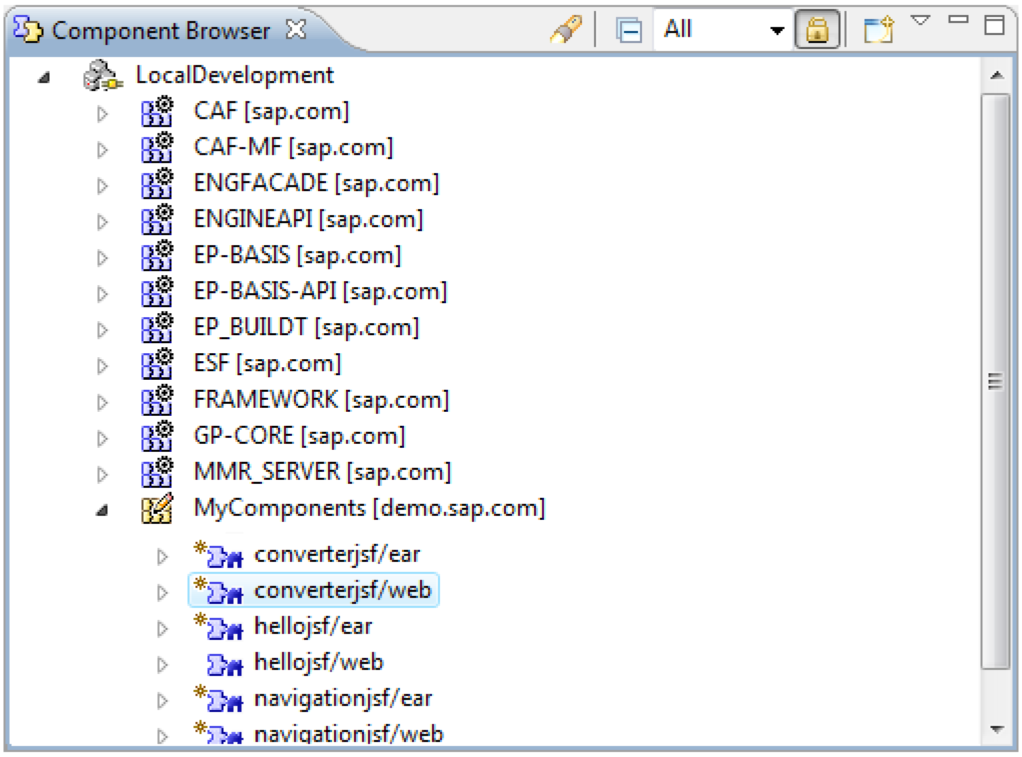
- Creating a converter class that implements the “javax.faces.convert.Converter” interface.
- Implementing the getAsObject() and getAsString() methods within your converter class.
- Registering your converter in faces-config.xml.
- Using the <f:converter> tag in your page view.

In addition you will create a custom validator class by

- Creating a validator class that implements the javax.faces.validator.Validator interface which will require a validate method.
- Registering your validator in faces-config.xml
- Using the <f:validator> tag in your page view.

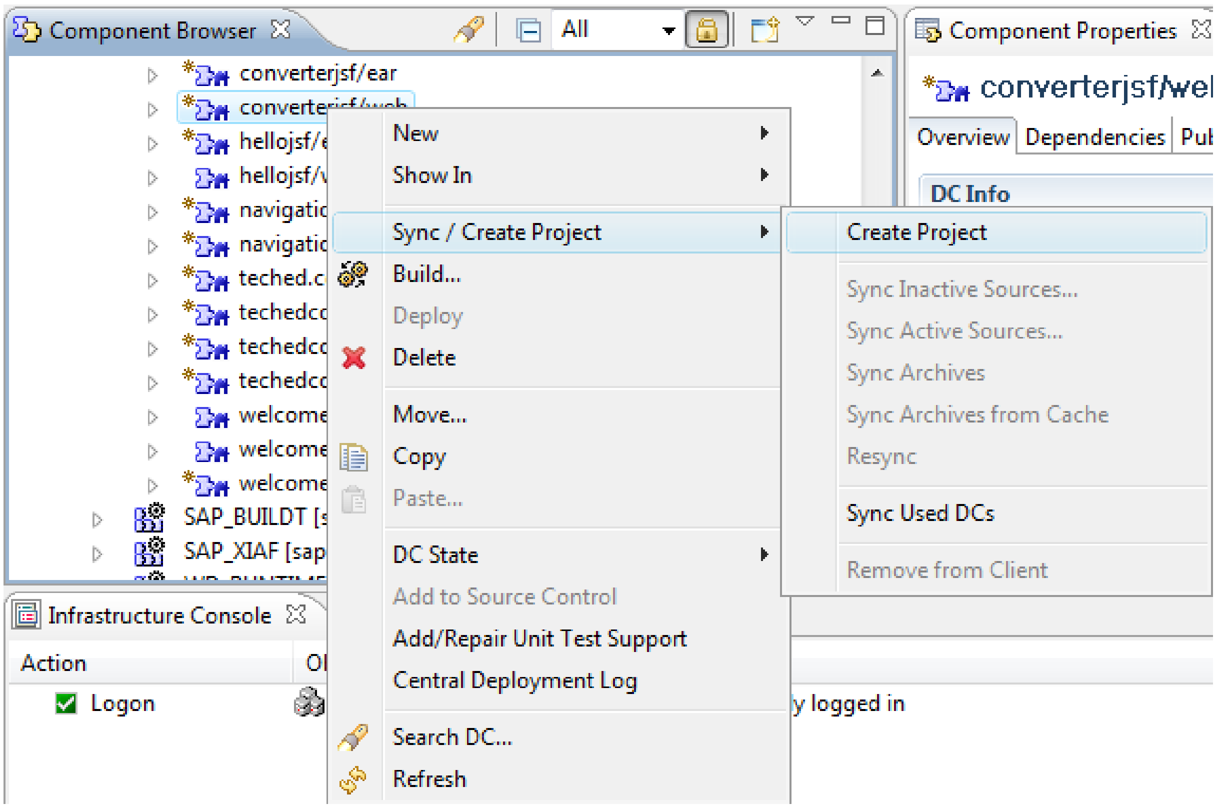
4.1 Tutorial Setup

1. Download the ZIP file *04_converterjsf_init.zip*, which contains the initial JSF project. Save it in a local directory.
2. Unzip the contents of the *04_converterjsf_init.zip* into the JDI workspace of the SAP NetWeaver Developer Studio under *LocalDevelopment* → *DCs* folder
3. Call the SAP NetWeaver Developer Studio and Open the *Development Infrastructure* perspective
4. Drill into LocalDevelopment -> MyComponents

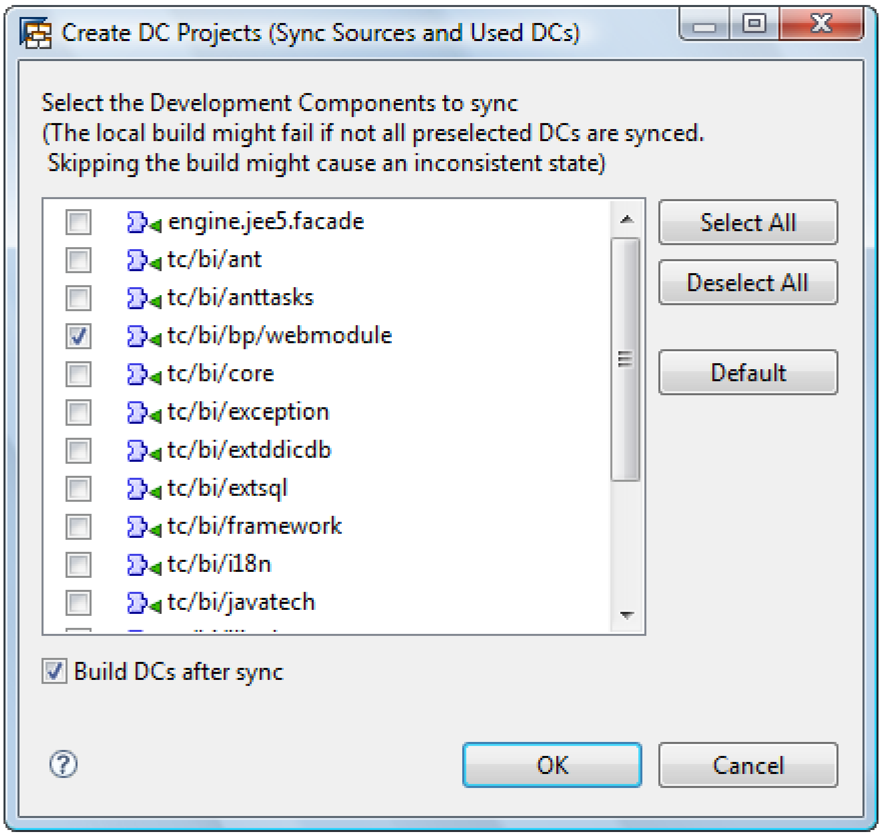


LocalDevelopment
CAF [sap.com]
CAF-MF [sap.com]
ENGFACADE [sap.com]
ENGINEAPI [sap.com]
EP-BASIS [sap.com]
EP-BASIS-API [sap.com]
EP_BUILDT [sap.com]
ESF [sap.com]
FRAMEWORK [sap.com]
GP-CORE [sap.com]
MMR_SERVER [sap.com]
MyComponents [demo.sap.com]
converterjsf/ear
converterjsf/web
hellojsf/ear
hellojsf/web
navigationjsf/ear
navigationjsf/web

5. Right click on the converterjsf/web component and select Sync / Create Project → *Create Project*



6. In the *Create DC Projects* window select **`tc/bi/bp/webmodule`** and click the *OK* button

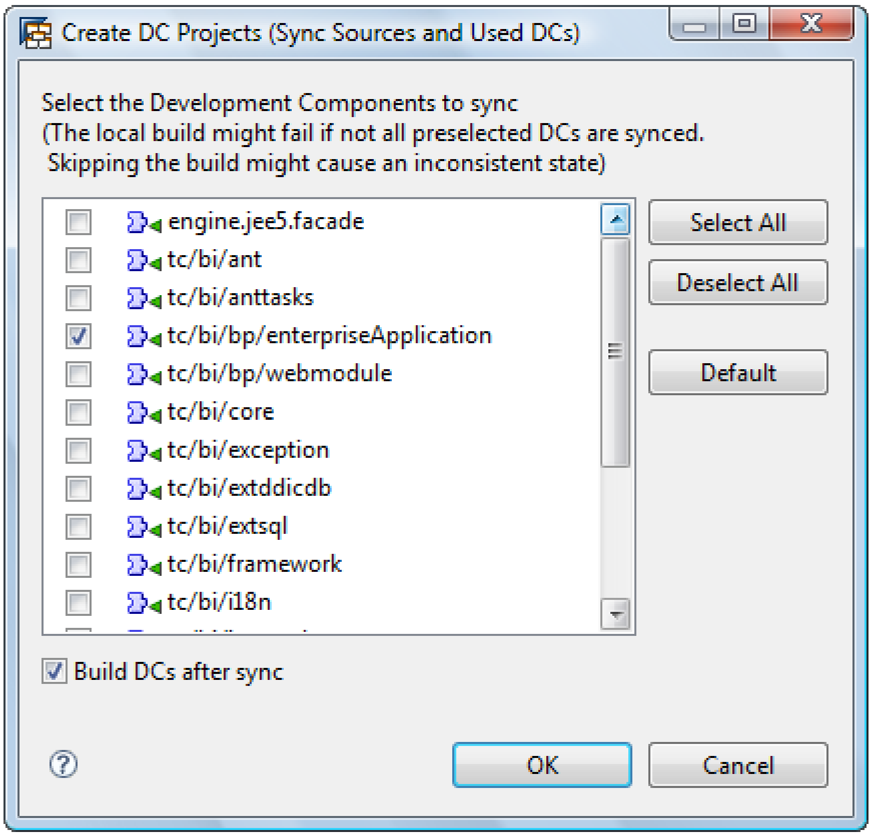


Select the Development Components to sync
(The local build might fail if not all preselected DCs are synced.
Skipping the build might cause an inconsistent state)

- [ ] engine.jee5.facade
- [ ] tc/bi/ant
- [ ] tc/bi/anttasks
- [x] tc/bi/bp/webmodule
- [ ] tc/bi/core
- [ ] tc/bi/exception
- [ ] tc/bi/extddicdb
- [ ] tc/bi/extsql
- [ ] tc/bi/framework
- [ ] tc/bi/i18n
- [ ] tc/bi/javatech

- [x] Build DCs after sync

7. Right click on the converterjsf/ear component and select Sync / Create Project → *Create Project*
8. In the *Create DC Projects* window select **`tc/bi/bp/enterpriseapplication`** and click the *OK* button

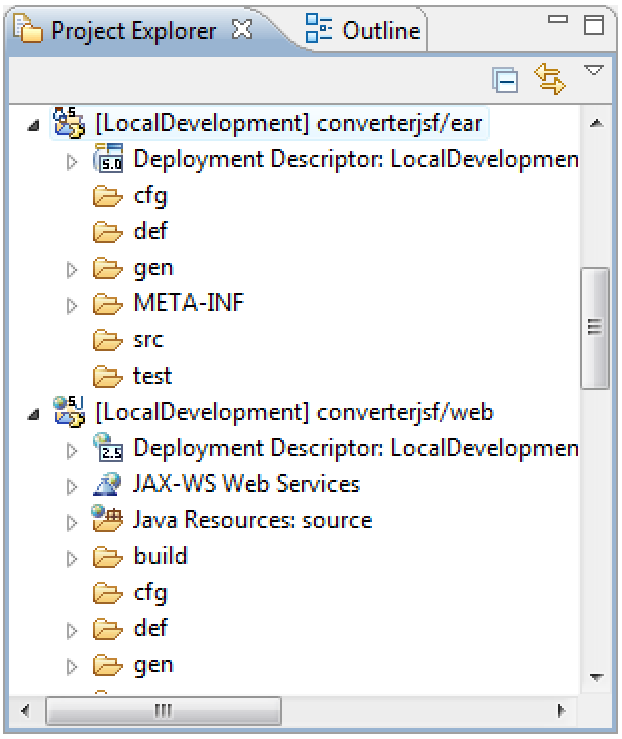


Select the Development Components to sync
(The local build might fail if not all preselected DCs are synced.
Skipping the build might cause an inconsistent state)

- [ ] engine.jee5.facade
- [ ] tc/bi/ant
- [ ] tc/bi/anttasks
- [x] tc/bi/bp/enterpriseApplication
- [ ] tc/bi/bp/webmodule
- [ ] tc/bi/core
- [ ] tc/bi/exception
- [ ] tc/bi/extddicdb
- [ ] tc/bi/extsql
- [ ] tc/bi/framework
- [ ] tc/bi/i18n

- [x] Build DCs after sync

9. Switch to the Java EE perspective. The *Development Components* have been imported in the *Project Explorer* window



```
[LocalDevelopment] converterjsf/ear
    Deployment Descriptor: LocalDevelopmen
    cfg
    def
    gen
    META-INF
    src
    test
[LocalDevelopment] converterjsf/web
    Deployment Descriptor: LocalDevelopmen
    JAX-WS Web Services
    Java Resources: source
    build
    cfg
    def
    gen
```

4.2 Extend Product Offer Tutorial

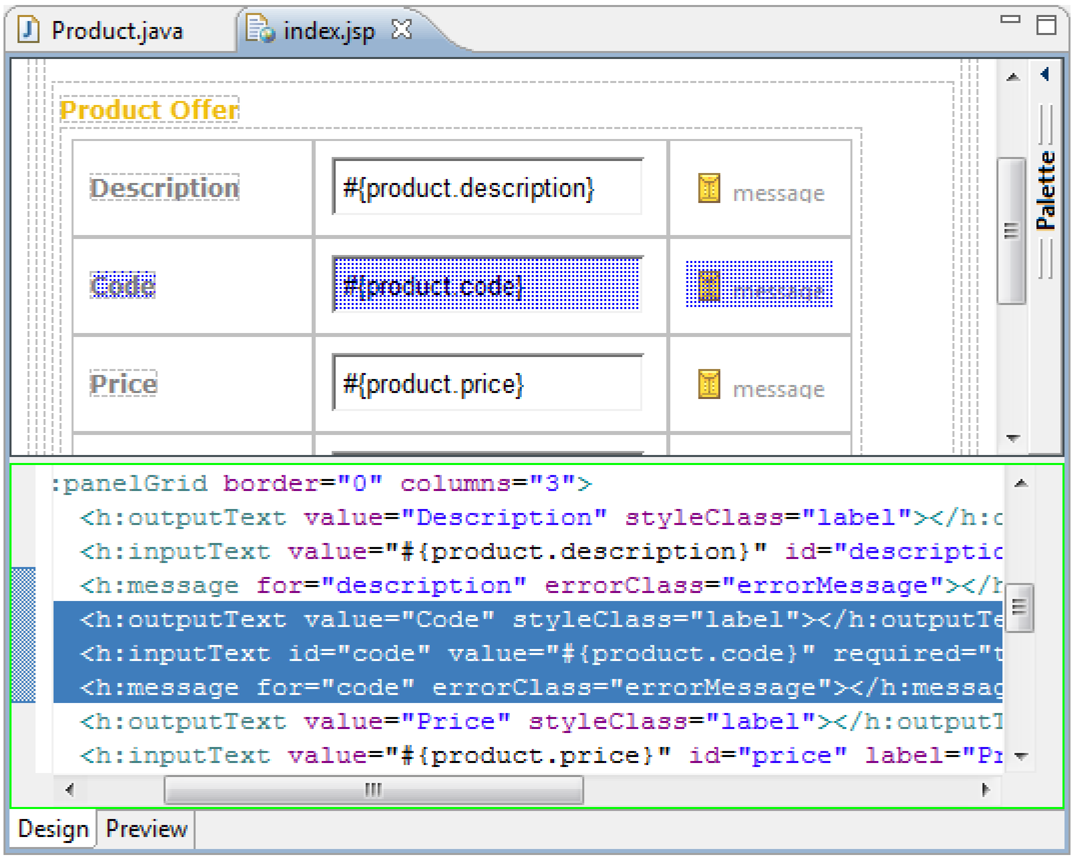
1. From the context menu of the *Java Resources: source* folder in the *Web Module* project open the Product.java class
2. Declare the *code* attribute and generate the corresponding *Getter* and *Setter* methods shown in the following code.

```
private String code;

…

public String getCode() {

   return code;

}


public void setCode(String code) {

   this.code = code;

}
```

3. In the *index.jsp* page place the following UI elements in the second row of the *Panel Grid* UI element.

Property	Value
OutputText UI element in the UI-element *PanelGrid*	
value	Code
styleClass	label
InputText UI element	
Id	code
value	#{product.code}
label	Code
required	true
Message UI element	
for	code
errorClass	errorMessage

4. Result of *index.jsp* page



Product Offer

Description	#{product.description}	message
Code	#{product.code}	message
Price	#{product.price}	message

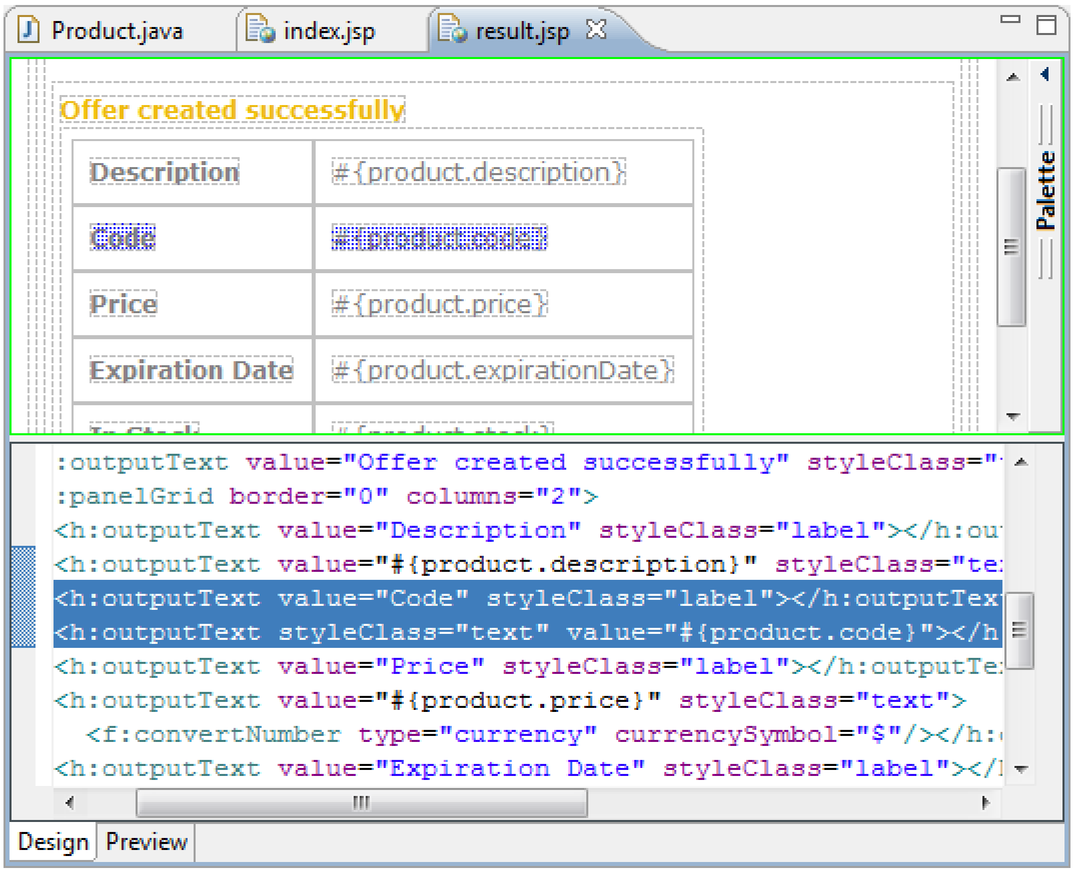
```
:panelGrid border="0" columns="3">
  <h:outputText value="Description" styleClass="label"></h:c
  <h:inputText value="#{product.description}" id="descriptic
  <h:message for="description" errorClass="errorMessage"></h
  <h:outputText value="Code" styleClass="label"></h:outputTe
  <h:inputText id="code" value="#{product.code}" required="t
  <h:message for="code" errorClass="errorMessage"></h:messag
  <h:outputText value="Price" styleClass="label"></h:outputT
  <h:inputText value="#{product.price}" id="price" label="Pr
```

5. Save the changes you made

6. In the *result.jsp* page place the following UI elements in the second row of the *Panel Grid* UI element.

Property	Value
OutputText UI element in the UI-element *PanelGrid*	
value	Code
styleClass	label
OutputText UI element in the UI-element *PanelGrid*	
value	#{product.code}
styleClass	text

7. Result of *result.jsp* page



Offer created successfully

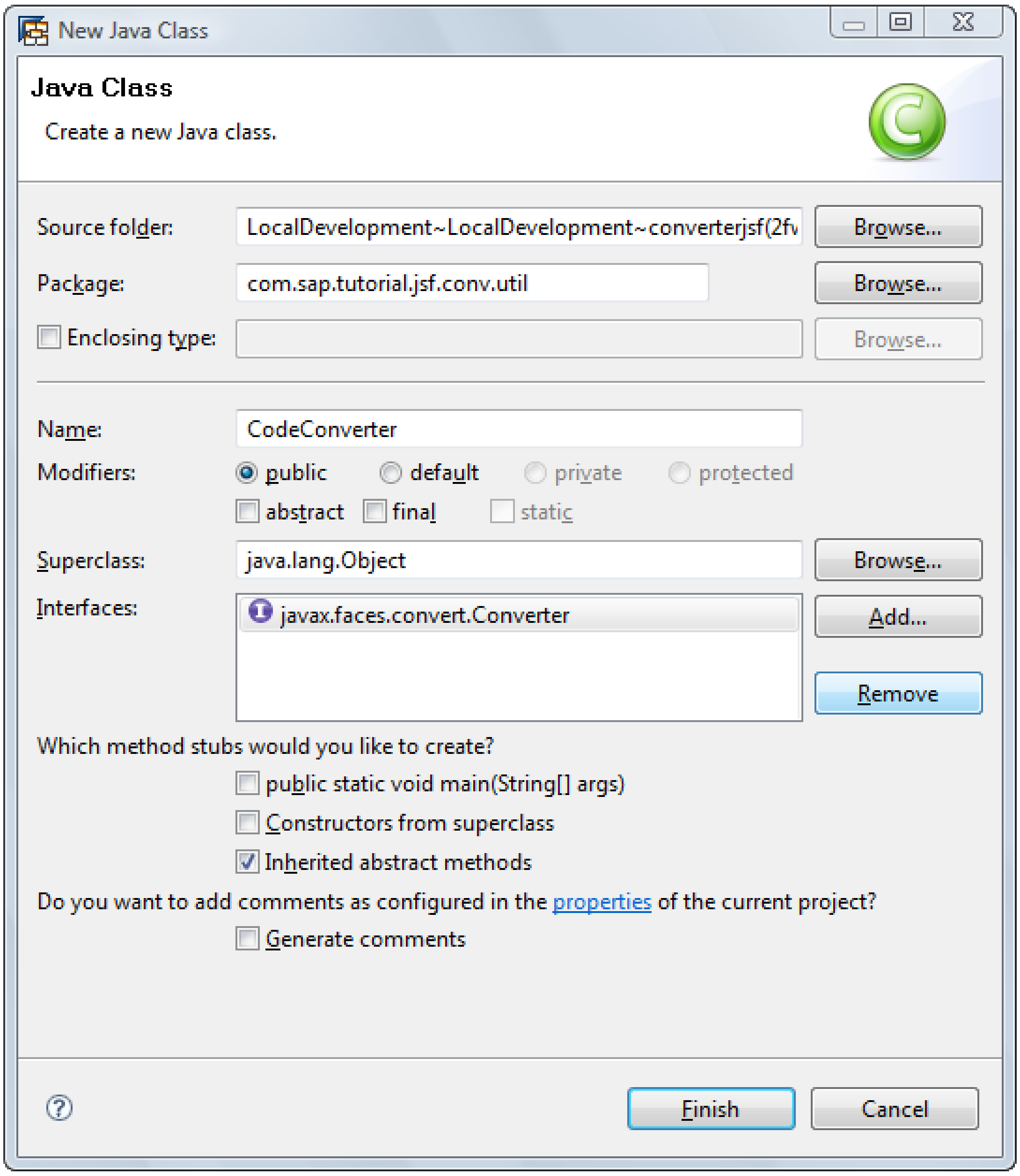
Description	#{product.description}
Code	#{product.code}
Price	#{product.price}
Expiration Date	#{product.expirationDate}

```
:outputText value="Offer created successfully" styleClass="
:panelGrid border="0" columns="2">
<h:outputText value="Description" styleClass="label"></h:ou
<h:outputText value="#{product.description}" styleClass="te
<h:outputText value="Code" styleClass="label"></h:outputTex
<h:outputText styleClass="text" value="#{product.code}"></h
<h:outputText value="Price" styleClass="label"></h:outputTe
<h:outputText value="#{product.price}" styleClass="text">
  <f:convertNumber type="currency" currencySymbol="$"/></h:
<h:outputText value="Expiration Date" styleClass="label"></
```

8. Save the changes you made

4.3 Implement Custom Converter Class

1. Create a converter class that implements the *javax.faces.convert.Converter* interface. From the context menu of the *Java Resources: source* folder in the *Web Module* project create a Java class. Enter **`CodeConverter`** in the *Name* field, **`com.sap.tutorial.jsf.conv.util`** in the *Package* field and add **`javax.faces.convert.Converter`** in the *Interfaces* field.


New Java Class

Java Class

Create a new Java class.

Source folder:	LocalDevelopment~LocalDevelopment~converterjsf(2fv	Browse...
Package:	com.sap.tutorial.jsf.conv.util	Browse...
☐ Enclosing type:		Browse...
Name:	CodeConverter	
Modifiers:	● public ○ default ○ private ○ protected ☐ abstract ☐ final ☐ static	
Superclass:	java.lang.Object	Browse...
Interfaces:	javax.faces.convert.Converter	Add... / Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

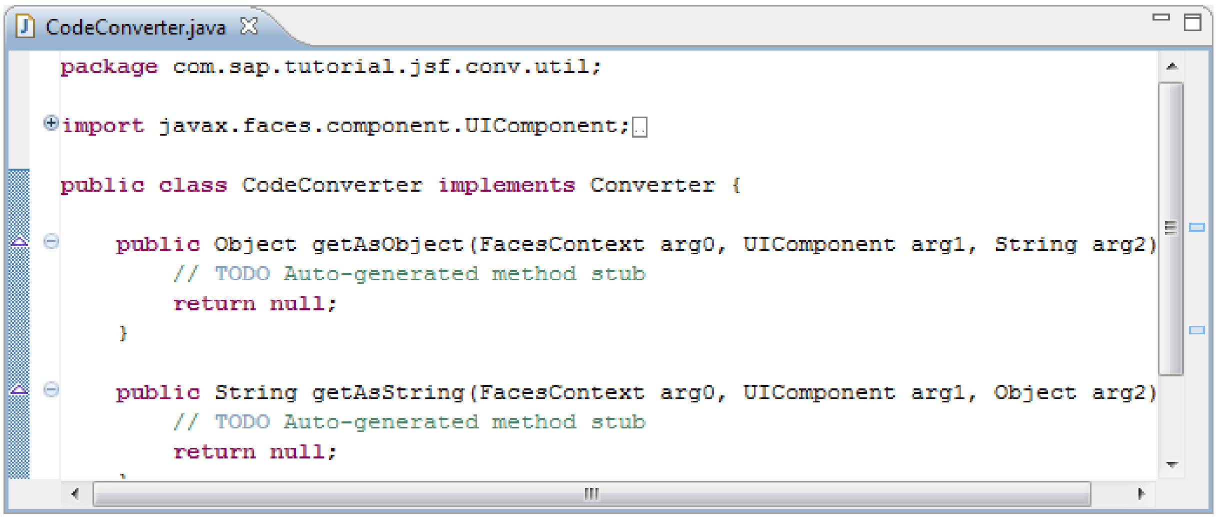
Do you want to add comments as configured in the properties of the current project?

☐ Generate comments

2. The converter class will be created. A converter must implement the Converter interface, which has the following two methods: getAsObject and getAsString.

   Note

   The *getAsObject* method converts a string into an object of the desired type, throwing a *ConverterException* if the conversion cannot be carried out. This method is called when a string is submitted from the client, typically in a text field. The *getAsString* method converts an object into a string representation to be displayed in the client interface.


CodeConverter.java

```
package com.sap.tutorial.jsf.conv.util;

import javax.faces.component.UIComponent;

public class CodeConverter implements Converter {

    public Object getAsObject(FacesContext arg0, UIComponent arg1, String arg2)
        // TODO Auto-generated method stub
        return null;
    }

    public String getAsString(FacesContext arg0, UIComponent arg1, Object arg2)
        // TODO Auto-generated method stub
        return null;
```

3. The *getAsObject* method in the *CodeConverter* class checks whether the product code has an invalid character. If it finds an invalid character, it throws a *ConverterException*. Implement the *getAsObject* by entering the following code in its body:

   Important

   Notice that the *ConverterException* method receives a *FacesMessage* object. The *FacesMessage* object contains the summary and detail messages that can be displayed with message tags.

```
public Object getAsObject(FacesContext arg0, UIComponent arg1, String arg2)
{
      int i = 0;
      boolean foundInvalidCharacter = false;
      String errorMessage = new String();
      if (arg2 == null){
         return null;
      }
      arg2 = arg2.replace("-", "").trim();
      StringBuilder builder = new StringBuilder(arg2);
      while (i < builder.length() && !foundInvalidCharacter) {
         char ch = builder.charAt(i);
         if (Character.isLetter(ch) || Character.isDigit(ch))
            i++;
         else if (Character.isWhitespace(ch))
            builder.deleteCharAt(i);
         else {
            foundInvalidCharacter = true;
            errorMessage = "Invalid character found '" + ch + "'";
         }
```

```
        }

        if (foundInvalidCharacter) {
            FacesMessage message = new FacesMessage();
            message.setDetail("Code: Convertion error: "+ errorMessage);
            message.setSummary("Code: Convertion error: " + errorMessage);
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ConverterException(message);
        }

        return builder.toString();
    }
```

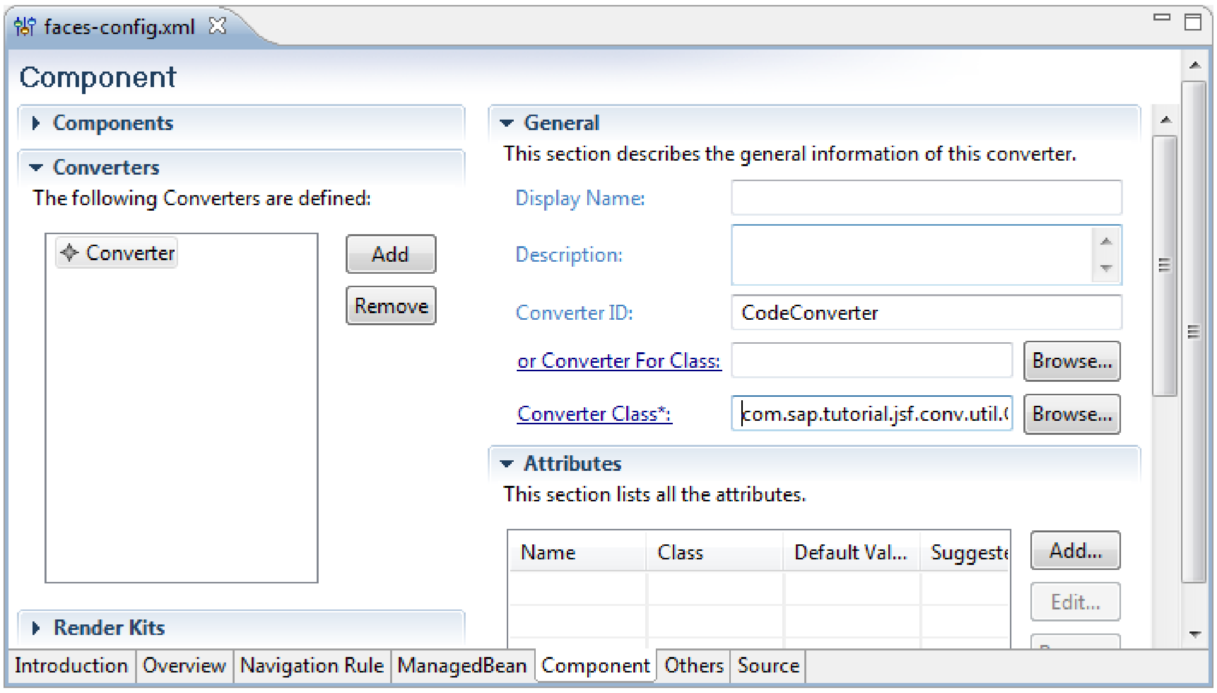
4. The *getAsString* method in the *CodeConverter* class allows users to enter the code without any special characters and it will format it using a dash to separate the characters the first three characters. Implement the *getAsString* method by entering the following code in its body

```
public String getAsString(FacesContext arg0, UIComponent arg1, Object arg2)
{
        if (arg2 == null){
            return null;
        }

        String code = arg2.toString();
        if (code.length()>=3){
            StringBuffer formattedCode = new StringBuffer();
            String part1 = code.substring(0,3);
            String part2 = code.substring(3,code.length());
            formattedCode.append(part1.toUpperCase());
            formattedCode.append("-");
            formattedCode.append(part2);
            return formattedCode.toString();
        } else{
            return code;
        }
    }
```

5. To register your converter in faces-config.xml, drill into the Web Module project, in the *WebContent* → *WEB-INF* folder and open the *faces-config.xml* file
6. Go to the *Component* tab, select the *Converters* section and click the *Add* button
7. Enter **`CodeConverter`** in the *Converter ID* field and **`com.sap.tutorial.jsf.conv.util.CodeConverter`** in the *Converter Class* field.

   Note

   The *Converter ID* field is a symbolic ID that you register with the JSF application



The following Converters are defined:

General

This section describes the general information of this converter.

Display Name:

Description:

Converter ID: CodeConverter

or Converter For Class:

Converter Class*: com.sap.tutorial.jsf.conv.util.(

Attributes

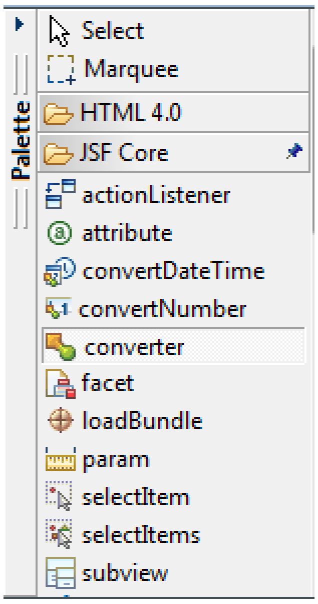
This section lists all the attributes.

Name	Class	Default Val...	Suggeste

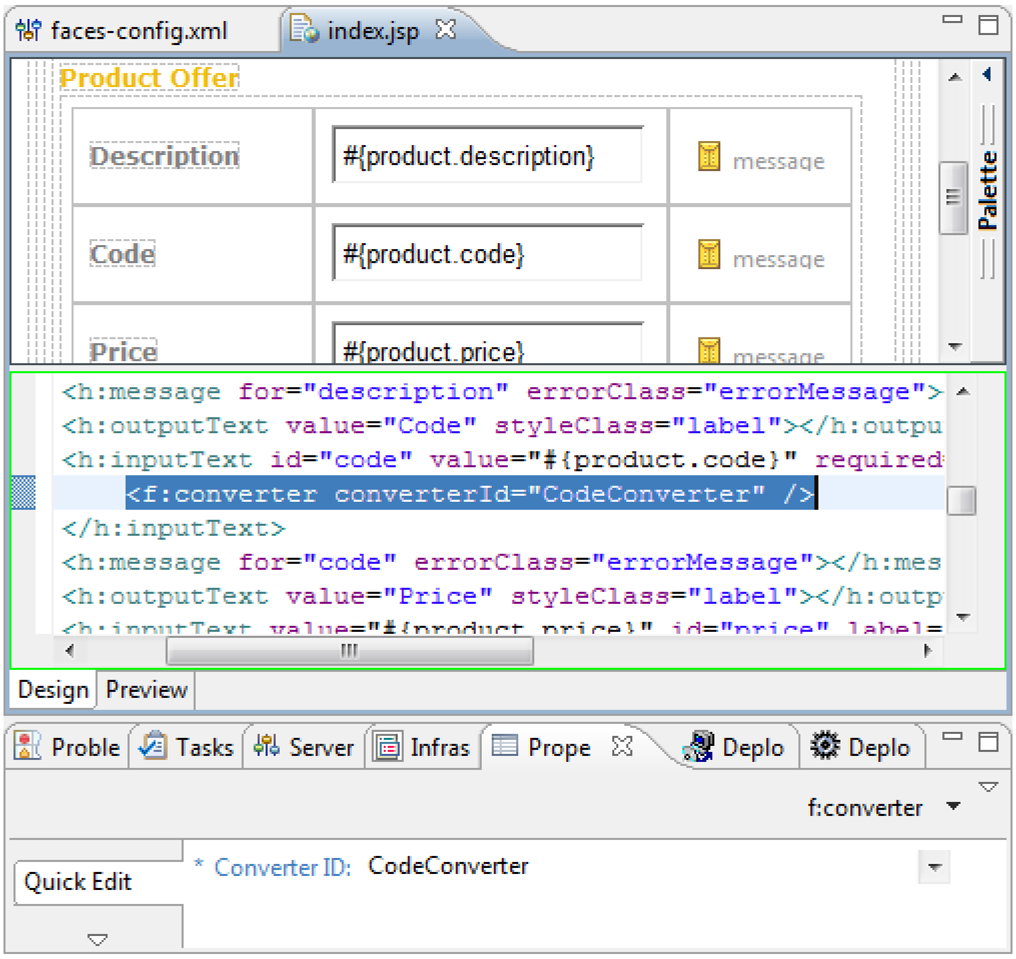
8. Save the changes you made
9. The following XML code is added automatically between the <faces-config … > </faces-config> tags

```
<converter>
   <converter-id>CodeConverter</converter-id>
   <converter-class>
      com.sap.tutorial.jsf.conv.util.CodeConverter
   </converter-class>
</converter>
```

10. In the *index.jsp* page click the *JSF Core* toolset in the Palette, this will show all the elements available within it



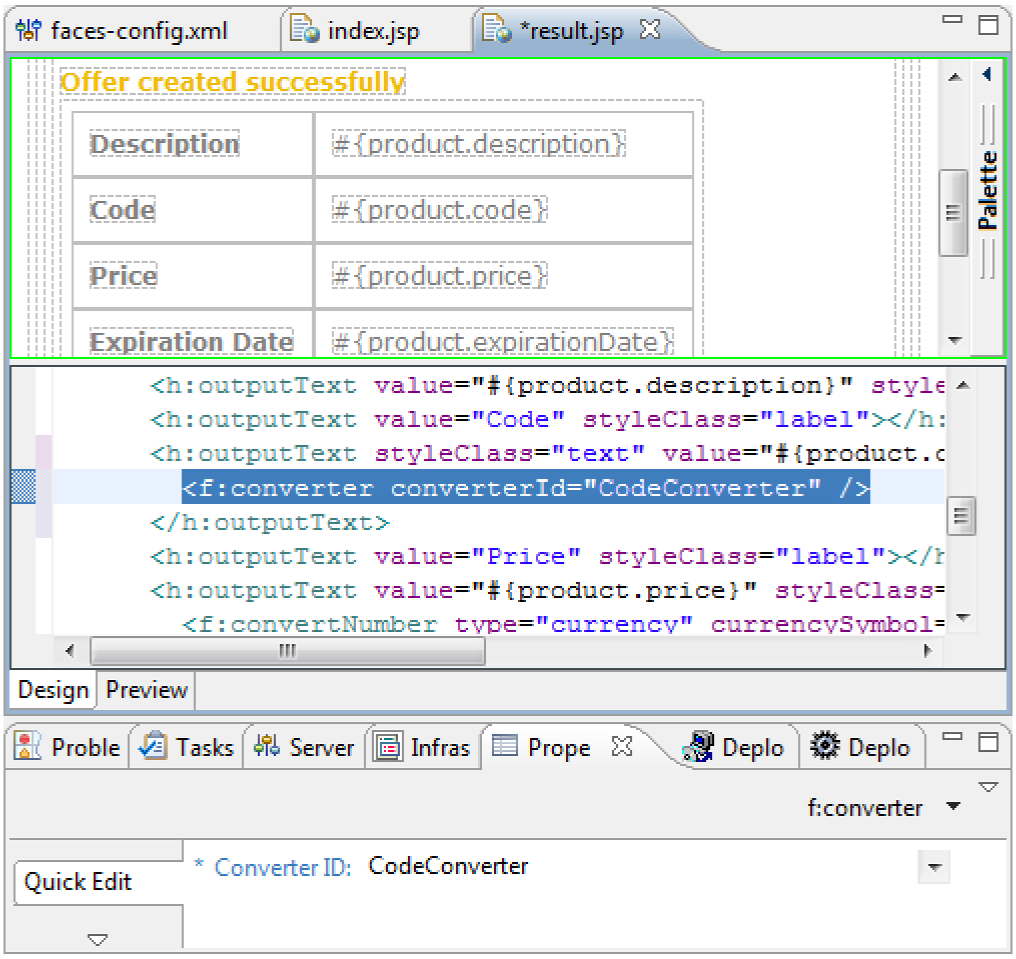
11. Place the *converter* element (found in the *JSF Core* elements) between the *code InputText* element (<h:inputText>… </h:inputText> tags) and set the *Converter ID* property to **`CodeConverter`**



```
<h:message for="description" errorClass="errorMessage">
<h:outputText value="Code" styleClass="label"></h:outpu
<h:inputText id="code" value="#{product.code}" required
    <f:converter converterId="CodeConverter" />
</h:inputText>
<h:message for="code" errorClass="errorMessage"></h:mes
<h:outputText value="Price" styleClass="label"></h:outp
<h:inputText value="#{product.price}" id="price" label=
```

* Converter ID: CodeConverter

12. Repeat the steps 10-11 in the *result.jsp* page



Offer created successfully

Description	#{product.description}
Code	#{product.code}
Price	#{product.price}
Expiration Date	#{product.expirationDate}

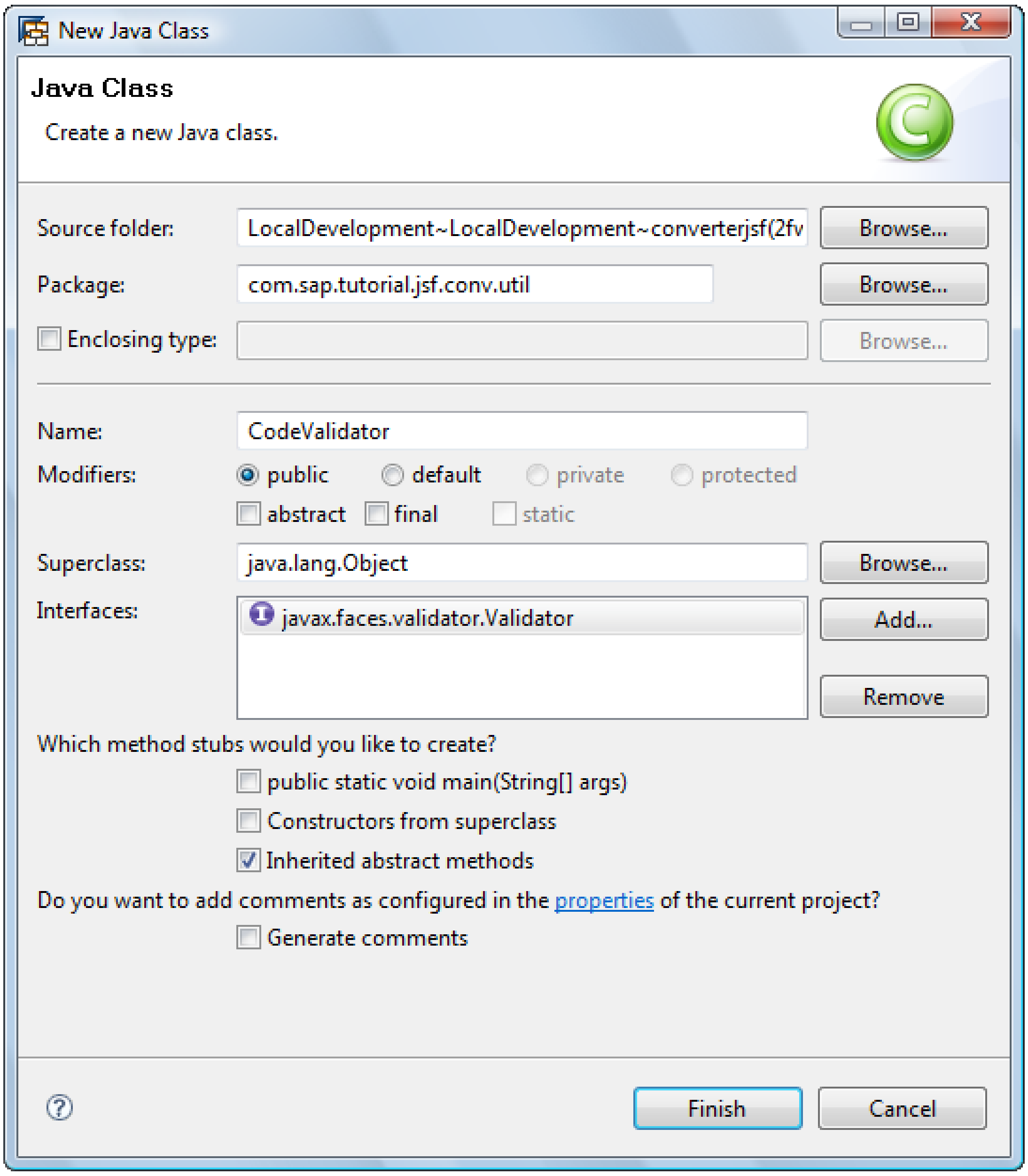
```
      <h:outputText value="#{product.description}" style
      <h:outputText value="Code" styleClass="label"></h:
      <h:outputText styleClass="text" value="#{product.c
        <f:converter converterId="CodeConverter" />
      </h:outputText>
      <h:outputText value="Price" styleClass="label"></h
      <h:outputText value="#{product.price}" styleClass=
        <f:convertNumber type="currency" currencySymbol=
```

* Converter ID: CodeConverter

13. Save the changes you made

4.4 Implement Custom Validator Class

1. Create a validator class that implements the *javax.faces.validator.Validator* interface. From the context menu of the *Java Resources: source* folder in the *Web Module* project create a Java class. Enter **`CodeValidator`** in the *Name* field, **`com.sap.tutorial.jsf.conv.util`** in the *Package* field and add **`javax.faces.validator.Validator`** in the *Interfaces* field.


New Java Class

Java Class

Create a new Java class.

Source folder: LocalDevelopment~LocalDevelopment~converterjsf(2fv

Package: com.sap.tutorial.jsf.conv.util

☐ Enclosing type:

Name: CodeValidator

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces: javax.faces.validator.Validator

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

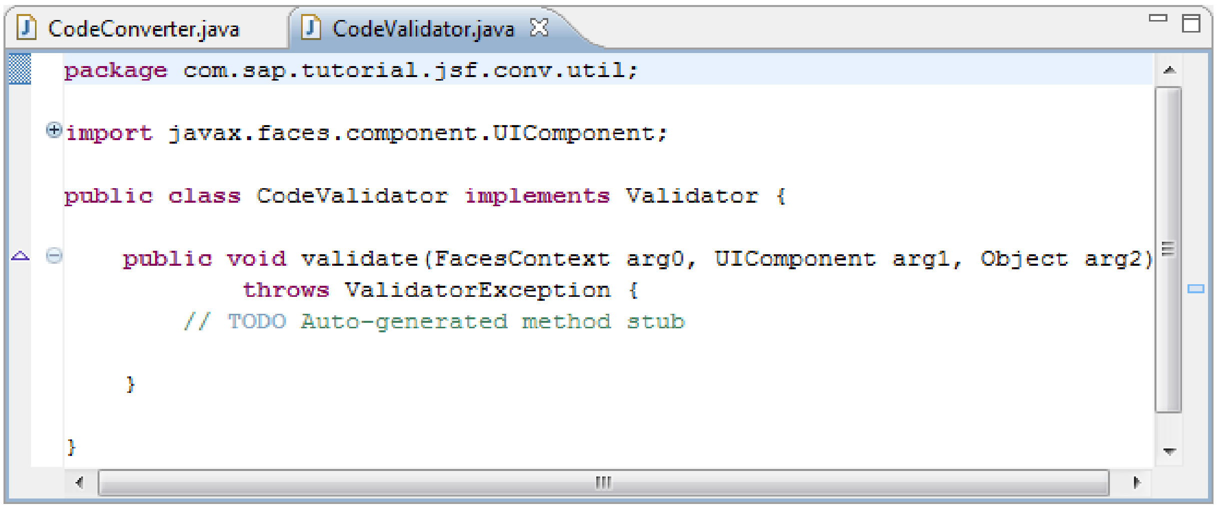
Do you want to add comments as configured in the properties of the current project?

☐ Generate comments

2. The validator class will be created. A validator must implement the Validator interface, which defines only one method: *validate*.

Note

The *validate* method validates the component to which this validator is attached. If there is a validation error, throw a *ValidatorException*



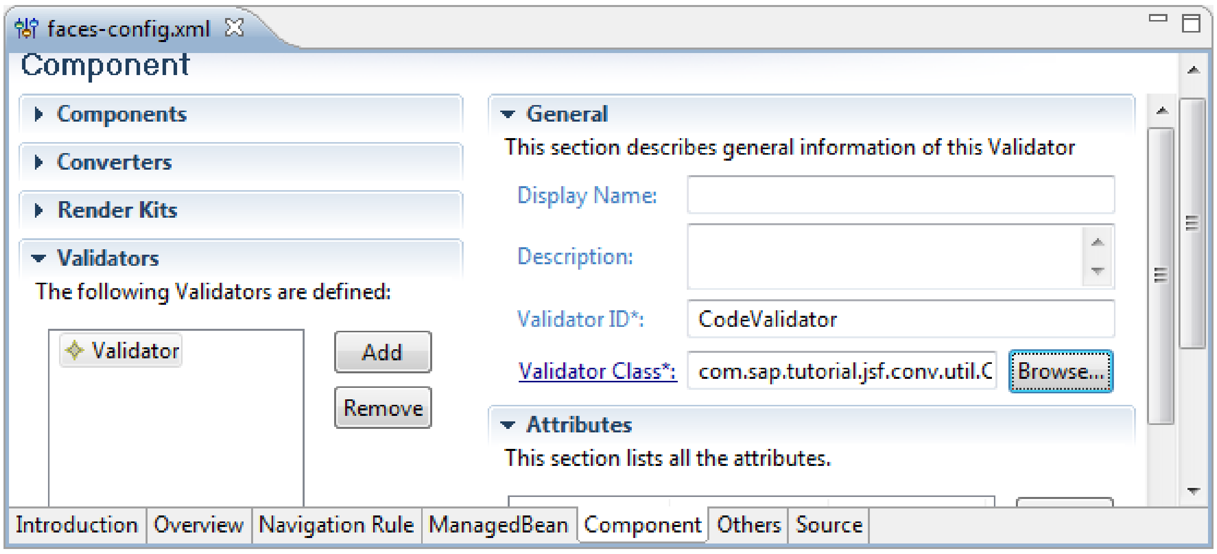
```
package com.sap.tutorial.jsf.conv.util;

import javax.faces.component.UIComponent;

public class CodeValidator implements Validator {

    public void validate(FacesContext arg0, UIComponent arg1, Object arg2)
            throws ValidatorException {
        // TODO Auto-generated method stub

    }

}
```

3. The *validate* method in the *CodeValidator* class checks whether the code has a valid length. It also checks if the code has three characters follow by a dash and four numbers (XXX-1234), if it is not the case, it will throw a *ValidatorException*. Implement the *validate* method by entering the following code in its body

```
public void validate(FacesContext arg0, UIComponent arg1, Object arg2)
      throws ValidatorException {
   int i = 0;
   boolean foundError = false;
   String errorMessage = new String();

   String code = arg2.toString();
   code = code.replace("-", "").trim();
   StringBuilder builder = new StringBuilder(code);
   if (builder.length() == 7) {
      while (i < builder.length() && !foundError) {
         char ch = builder.charAt(i);
         if (Character.isLetter(ch) && i < 3)
            i++;
         else if (Character.isDigit(ch) && i >= 3)
            i++;
         else if (Character.isWhitespace(ch))
            builder.deleteCharAt(i);
         else {
            foundError = true;
            errorMessage = "Invalid code. It should be 'XYZ1234' or
'XYZ-1234'";
         }
```

```
            }
        } else{
            foundError = true;
            errorMessage = "Invalid length '" + builder.length() + "'. Lenght
        allowed is 7 or 8";
        }

        if (foundError) {
            FacesMessage message = new FacesMessage();
            message.setDetail("Code: Validation error: "+ errorMessage);
            message.setSummary("Code: Validation error: " + errorMessage);
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(message);
        }

    }
```

4. To register your converter in faces-config.xml, drill into the Web Module project, in the *WebContent* → *WEB-INF* folder and open the *faces-config.xml* file
5. Go to the *Component* tab, select the *Validators* section and click the *Add* button
6. Enter **`CodeValidator`** in the *Validator ID* field and **`com.sap.tutorial.jsf.conv.util.CodeValidator`** in the *Validator Class* field.



The following Validators are defined:

Validator

General

This section describes general information of this Validator

Display Name:

Description:

Validator ID*: CodeValidator

Validator Class*: com.sap.tutorial.jsf.conv.util.C

Attributes

This section lists all the attributes.

7. Save the changes you made
8. The following XML code is added automatically between the <faces-config … > </faces-config> tags

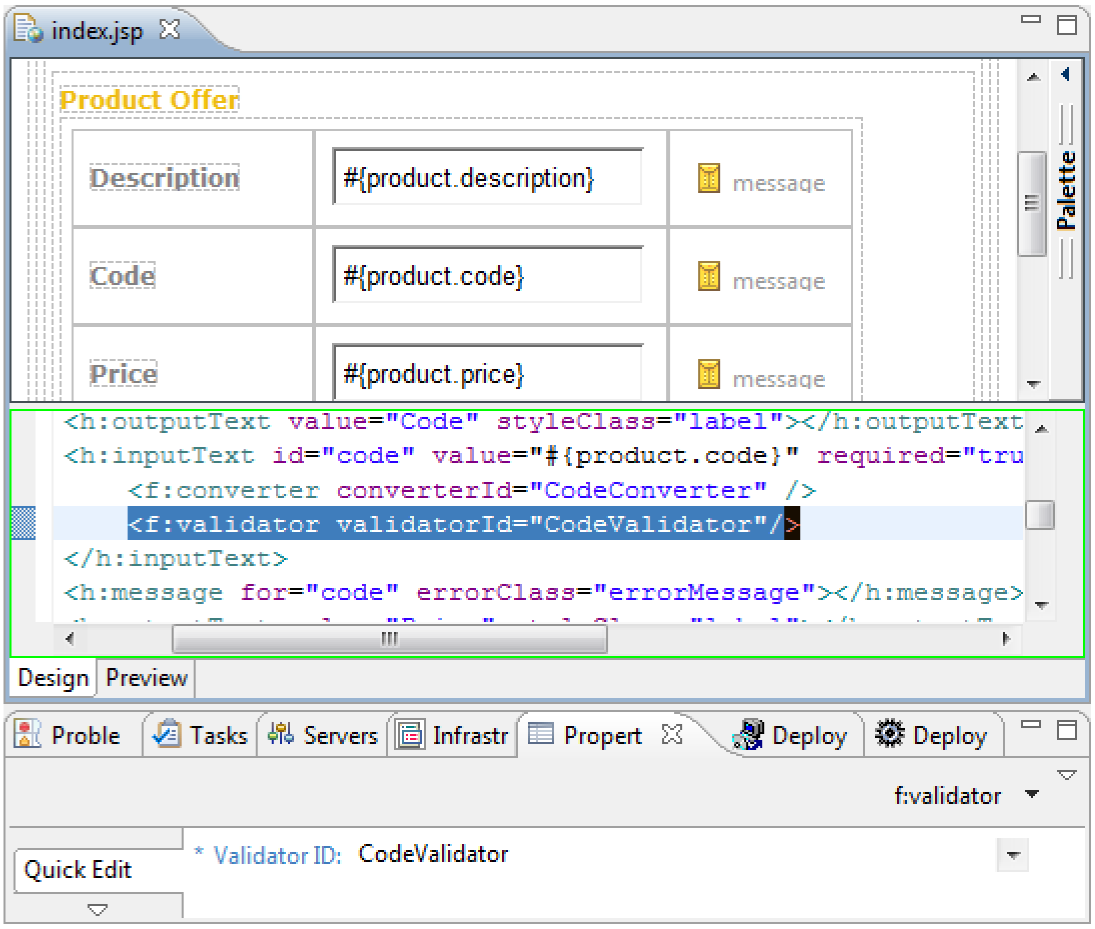
```
<validator>
```

```
        <validator-id>CodeValidator</validator-id>
        <validator-class>
           com.sap.tutorial.jsf.conv.util.CodeValidator
        </validator-class>
    </validator>
```

9. In the index.jsp, place the *validator* element (found in the *JSF Core* elements) between the *code InputText* element (<h:inputText>… </h:inputText> tags) and set the *Validator ID* property to **`CodeValidator`**

   Note

   The ValidatorID specified for f:validator must correspond to a validator ID specified in the *faces-config.xml* file (step 6)



```
<h:outputText value="Code" styleClass="label"></h:outputText
<h:inputText id="code" value="#{product.code}" required="tru
    <f:converter converterId="CodeConverter" />
    <f:validator validatorId="CodeValidator"/>
</h:inputText>
<h:message for="code" errorClass="errorMessage"></h:message>
```

* Validator ID: CodeValidator

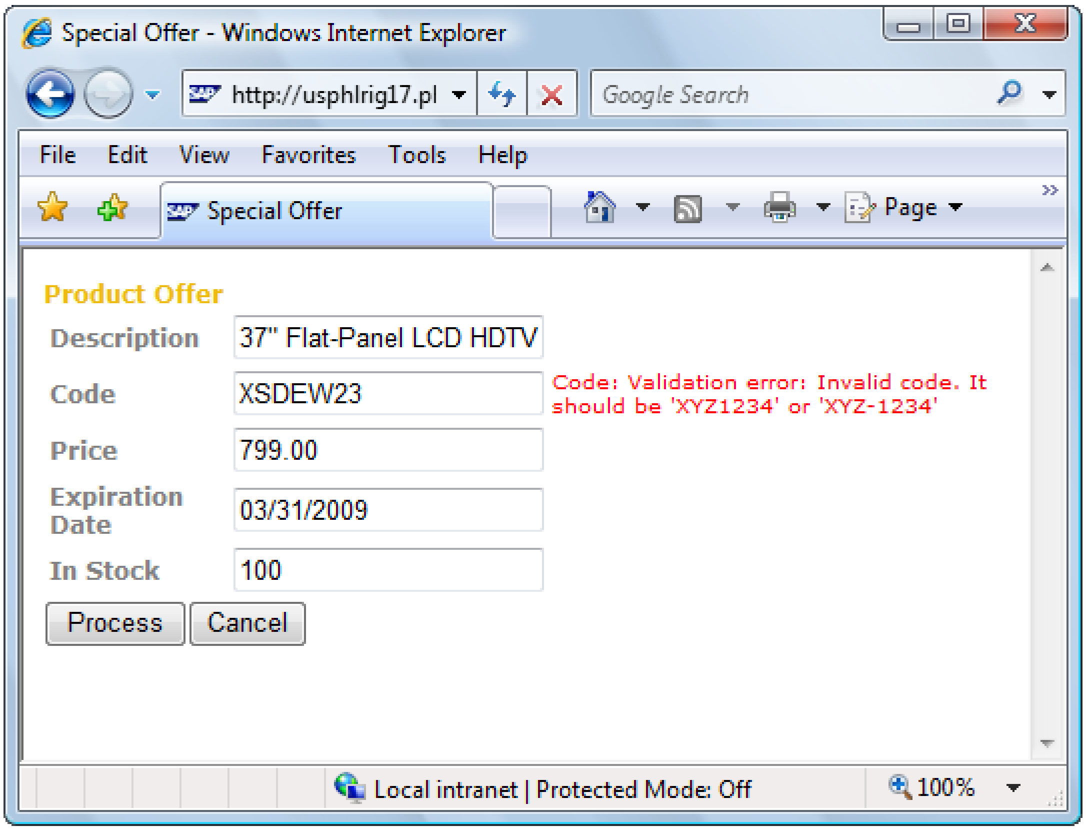
10. Save the changes you made

4.5 Build, Deploy and Run your application

1. Build and deploy the application.
2. Run the application using the simplified URL:

http://<servername>:<httpport>/converterjsf/faces/index.jsp

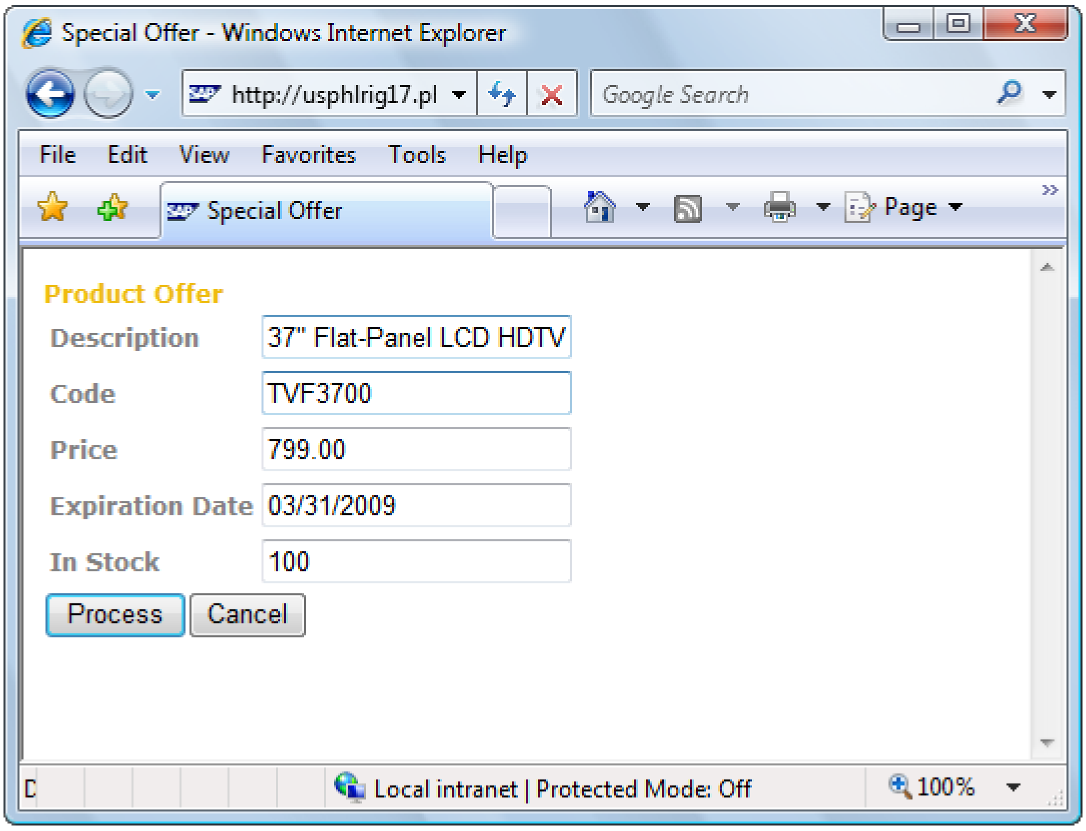
3. Results:


Special Offer - Windows Internet Explorer
http://usphlrig17.pl
Google Search
File Edit View Favorites Tools Help
Special Offer
Page
Process
Cancel
Local intranet | Protected Mode: Off
100%

Product Offer

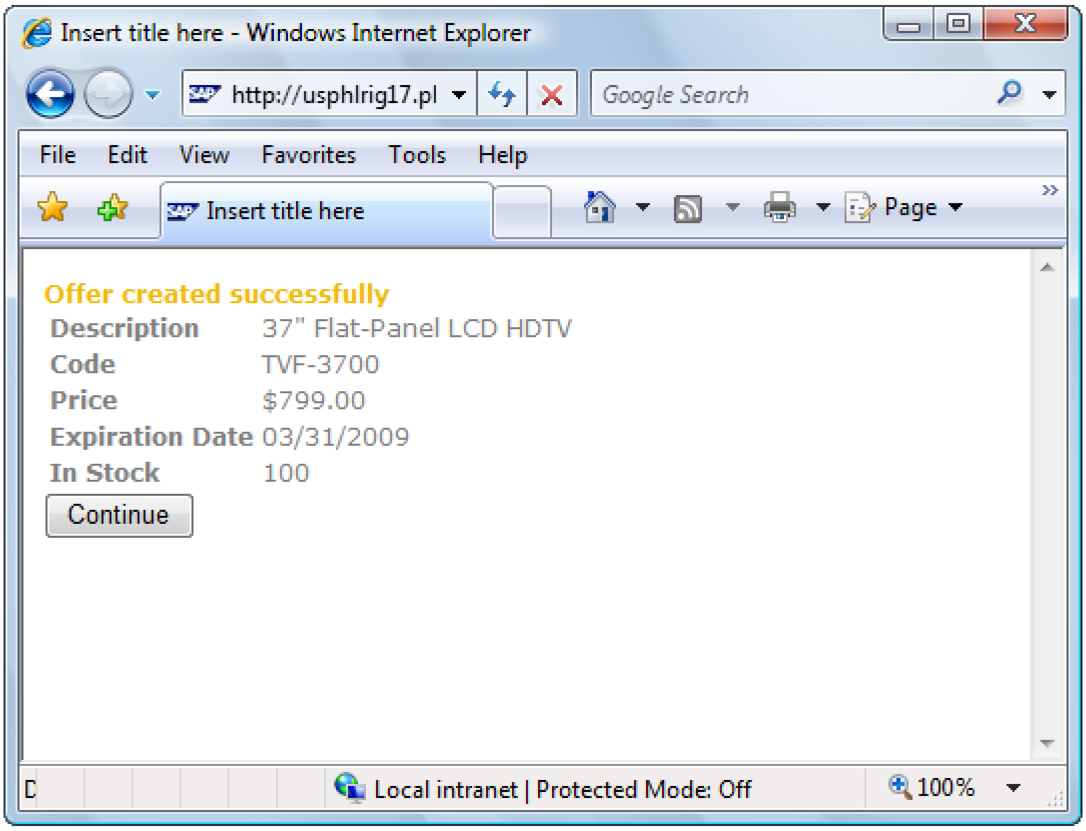
Description	37" Flat-Panel LCD HDTV
Code	XSDEW23
Price	799.00
Expiration Date	03/31/2009
In Stock	100

Code: Validation error: Invalid code. It should be 'XYZ1234' or 'XYZ-1234'


Special Offer - Windows Internet Explorer
http://usphlrig17.pl
Google Search
File Edit View Favorites Tools Help
Special Offer
Page
Process
Cancel
Local intranet | Protected Mode: Off
100%

Product Offer

Description	37" Flat-Panel LCD HDTV
Code	TVF3700
Price	799.00
Expiration Date	03/31/2009
In Stock	100


Insert title here - Windows Internet Explorer
http://usphlrig17.pl
Google Search
File Edit View Favorites Tools Help
Insert title here
Page
Local intranet | Protected Mode: Off
100%

Offer created successfully

Description	37" Flat-Panel LCD HDTV
Code	TVF-3700
Price	$799.00
Expiration Date	03/31/2009
In Stock	100

Continue

www.sdn.sap.com/irj/sdn/howtoguides

THE BEST-RUN BUSINESSES RUN SAP™