

Build Web sites with BPEL business processes

Skill Level: Intermediate

[Diana Lau \(dhmlau@ca.ibm.com\)](mailto:dhmlau@ca.ibm.com)

Software Developer

EMC

01 Jul 2004

Create a Web site that uses business processes to perform daily business operations. With this example, you use WebSphere® Studio Application Developer Integration Edition V5.1 to build a Web site that takes pizza orders. If the customer has a good credit history, the order is placed, and the time needed for pickup or delivery is calculated and shown on the confirmation page. If the customer has bad credit, an order cannot be placed and the customer is informed.

Section 1. Before you start

What is this tutorial about?

The IBM WebSphere Studio Application Developer Integration Edition, Version 5.1 has Business Process Execution Language (BPEL) support to orchestrate Web services to create business processes.

This tutorial explains how to create a Web site that uses business processes to perform daily business operations. Follow three major steps to build this example:

1. Create a Web site using Struts and JSP files.
2. Create Web services.
3. Create BPEL processes.

With this example, you build a Web site that takes pizza orders. If the customer has a good credit history, the order is placed, and the time needed for pickup or delivery is calculated and shown on the confirmation page. If the customer has bad credit, an order cannot be placed and the customer is informed.

You create two Web services: One service is for checking a customer's credit, and another service is for calculating the time needed to prepare the pizza. The whole procedure is captured in a BPEL business process.

Prerequisites

To run this example, you need to install WebSphere Studio Application Developer Integration Edition Version 5.1 ([Trial version available](#)).

Familiarity with this product is not required; however, some experience with WebSphere Studio or Eclipse is necessary.

Section 2. Create a Web site using Struts

Struts overview

Struts is an open source framework for building Java Web applications. It is based on the Model-View-Controller (MVC) design pattern. It includes ActionForms to collect inputs from users, ActionMappings to direct input to server-side Actions, and ActionForwards to select output pages.

This section is based on the *Ordering Pizza* Struts example found on the IBM developerWorks Web site (see [Resources](#)).

Create a Web project with Struts support and page template

1. From the Business Integration perspective, switch to the Web perspective.
2. Select **File --> New --> Web --> Dynamic Web Project**. A dynamic Web project is necessary, since it involves JSP files.
3. Name the Web project *PizzaWeb*.

4. Ensure the **configure advanced option** check box is selected. Click **Next**.
5. Enter *PizzaEAR* as the EAR project. Click **Next**.
6. Check the **Add Struts support** box. Continue selecting **Next** until you reach the Page Template page.
7. Check **use a default Page Template for the Web Site**, and select **B-03_blue.htpl** as the template. This template provides the navigation bar, header, and footer. Click **Finish**.
8. Create the index.html file with the B-03_blue.htpl template by selecting **File --> New --> HTML/XHTML file**.
9. Create the same template, this time with Struts tags. Copy and paste B-03_blue.htpl from the PizzaWeb\WebContent\theme directory, and rename it to *B-03_blue.jtpl*. Then add the tags shown below within the <head> tag.

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
```

You now have the template for the JSP files. You need the additional tags to run the Web page with Struts support. You can add these tags to the individual JSP files; however, they will be erased once the template is modified, and adding the tags to every file can be tedious.

Create JSP pages

Create two JSP pages:

- *placeOrder.jsp* lets users order the pizza
- *confirm.jsp*: indicates whether the order has been placed successfully.

Create placeOrder.jsp from the template:

1. Select **File --> New --> JSP File**. The new JSP file wizard launches.
2. Click **Next**.
3. Click **Finish**.

Use the Struts HTML tags palette to create an HTML form that contains the following elements:

Alternatively, you can add the following snippet to define the form:

```
<P>Order a Pizza<BR>
</P>
<html:form action="/placeOrder.do">

First Name: <html:text property="customer.firstname" />
<BR>Last Name: <html:text property=
"customer.lastname"/><br>
Email Address: <html:text property="customer.email"/><br>
Address: <html:text property="customer.address"/><br>
Credit Card Number:
<html:text property="customer.creditCardNum"/><br>
Size: <html:radio property="size" value="S"/>Small
<html:radio property="size" value="M"/>Medium
<html:radio property="size" value="L"/>Large

<BR>
<BR>Toppings: <br>
Pepperoni<html:checkbox property="topping(Pepperoni)"/><br>
Onion<html:checkbox property="topping(Onion)"/><br>
Mushroom<html:checkbox property="topping(Mushroom)"/><br>
Hot Pepper<html:checkbox property="topping(Hot Pepper)"/><br>
Bacon<html:checkbox property="topping(Bacon)"/><br>

<html:select property="type">
<html:option value="a">Delivery</html:option>
<html:option value="b">Pickup</html:option>
</html:select>

<html:submit/>
<html:reset/>
</html:form>
```

Finally, create a confirmation JSP file, confirm.jsp. This file reflects the inputs gathered in placeOrder.jsp.

```
<P>Thank you <%=request.getAttribute("firstname")%></P>
<P>Your <%=request.getAttribute("size")%> pizza with:<BR>
<%=request.getAttribute("toppings")%>

<BR>
will be available by:
<%=request.getAttribute("type")%> in 10 minutes
```

```
</P>
```

Web site navigation

The text *Highlighted* and *Normal* display on the navigation bar, header, and footer areas that you did not enter. This information comes from the template you used to create the HTML and JSP files.

Use the Web Site Navigation editor to create the structure of the Web site in a tree representation. The benefit of this step is that the Web site is easier to manage. You can generate a site map with a few mouse clicks. You can invoke this editor by double-clicking on **Web Site Navigation** under the Web project:

The main page, or welcome page -- index.html in this example -- is usually the root of the navigation tree. Drag index.html on the editor. Select **Edit --> Edit Navigation Label** to change the navigation label to *Home*.

Create the rest of the Web site structure by adding files below the root. Right-click on the **Home** node, and select **add existing page**. Press the **Add Other Pages** button to add placeOrder.jsp as a child.

Add confirm.jsp as the child of placeOrder.jsp. Edit the navigation label as appropriate.

When you are finished, you should see the following diagram in the editor. You can set properties on a particular node on the tree to indicate if the node is shown in the navigation bar and site map.

Create ActionForm: PlaceOrderForm.java

The ActionForm class is used to get the inputs from users in the forms. It also has validation and reset methods. The ActionForm class can invoke EJBs, establish JDBC connections, and, in this example, call the proxy to a Web service (which will be covered later in this tutorial).

1. Select **New --> Other --> Web --> Struts --> ActionForm class**. Click **Next**.
2. Enter the ActionForm class name as *PlaceOrderForm*. Click **Next**.
3. Expand the navigation tree to show the widgets in placeOrder.jsp, since this ActionForm is retrieving data from this JSP page.

4. Click **Next**.
5. Change the properties to look like those shown below. Ignore the error, and click **Finish**.
6. In the PlaceOrderActionForm class, change CustomerClass to be a public inner class, since this will be referenced later, outside the parent class. You can also move this class out to be a separate class for reusability. The declaration of the CustomerClass looks like:

```
public class CustomerClass
```

7. It is assumed that nothing is null; therefore, change the declarations to match this:

```
private java.util.HashMap toppings = new HashMap();  
private CustomerClass customer = new CustomerClass();
```

8. Change the reset method:

```
public void reset(ActionMapping mapping, HttpServletRequest request) {  
    // Reset values are provided as samples only. Change as appropriate.  
  
    toppings = new HashMap();  
    size = null;  
    type = null;  
    customer = new CustomerClass();  
}
```

9. You need separate methods to get and set the topping information, so add the following methods:

```
public void setTopping(String key, Object value)  
{  
    toppings.put(key, value);  
}  
public Object getTopping(String key)  
{  
    if (toppings.get(key)==null) return "no";  
    if (toppings.get(key).equals("on"))  
        return "yes";  
    else  
        return "no";  
}
```

Create the Action class: PlaceOrderAction

The Action class contains the business logic.

To create the Action class:

1. **File --> New --> Other --> Web --> Struts --> Action Class**
2. Name it `PlaceOrderAction`.
3. Click **Next**. Set up the mapping configuration.
4. For now, `PlaceOrderAction` only reflects part of the information that users enter. Add the snippet shown below, inside the try-catch block:

```
String firstname = placeOrderForm.getCustomer().getFirstname();
request.setAttribute("firstname", firstname);
String lastname = placeOrderForm.getCustomer().getLastname();
request.setAttribute("lastname", lastname);

String size = placeOrderForm.getSize();
if (size.equals("S"))
    request.setAttribute("size", "small");
else if (size.equals("M"))
    request.setAttribute("size", "medium");
else
    request.setAttribute("size", "large");

String email = placeOrderForm.getCustomer().getEmail();
request.setAttribute("email", email);

String address = placeOrderForm.getCustomer().getAddress();
request.setAttribute("address", address);

String type = placeOrderForm.getType();
if (type.equals("a"))
    request.setAttribute("type", "Delivery");
else
    request.setAttribute("type", "PickUp");

HashMap toppings = placeOrderForm.getToppings();
String toppingsString = "";
for (Iterator it= toppings.keySet().iterator(); it.hasNext();) {
    toppingsString += it.next() + ",";
}
toppingsString = toppingsString.substring(0, toppingsString.length()-1);
request.setAttribute("toppings", toppingsString);
```

Create a server and run a preview of the Web site

1. Right-click on the server view, select **New --> Server and Server Configuration**.

2. Select **WebSphere version 5.1 --> Integration Test Environment** as the server type. Click **Finish**.
 3. Right-click on **index.html** and select **Run on Server**.
 4. Select the server just created.
 5. From the Web page, click **Order a Pizza** to show the content of `placeOrder.jsp`.
 6. Enter the appropriate information in the form and click **Submit**. The result should look like this:
-

Section 3. Create a Web service

Introduction

In this section, you create two Web services. One service is for calculating the time needed to prepare the pizza for delivery or pick up, depending on the customer's preference in the order. You use the other Web service to check the customer's credit.

Create a Service Project

1. Switch to the Business Integration perspective.
2. Select **New --> Service Project**.
3. Name it *PlaceOrderSP*. Click **Finish**.

Create a Java class for implementation

You can use different programming languages to implement Web services. For this example, use Java language.

Create the Java class `calculate.time.CalculateTime` with the `calculateTime` method:

```
public int calculateTime(order.definition.Order order)
{
    int timeNeeded = 0;
    switch (order.getSize()) {
        case Order.SIZE_LARGE :
            timeNeeded = 20;
            break;
        case Order.SIZE_MEDIUM :
            timeNeeded = 10;
        default :
            timeNeeded = 5;
            break;
    }
    timeNeeded += order.getToppings().length() * 2;
    return timeNeeded;
}
```

The complex type `order.definition.Order` looks like the following sample. It implements `java.io.Serializable` so that it can be serialized at run time.

```
public class Order implements Serializable {

    //default pizza size
    private int size = 1; //1 - small, 2-medium, 3-large

    //different pizza size expressed in integer
    public static final int SIZE_SMALL = 1;
    public static final int SIZE_MEDIUM = 2;
    public static final int SIZE_LARGE = 3;

    //different pizza size expressed in string
    public static final String SIZE_SMALL_STRING = "small";
    public static final String SIZE_MEDIUM_STRING = "medium";
    public static final String SIZE_LARGE_STRING = "large";

    // order status code
    public static final int ORDER_STATUS_PENDING = 0;
    public static final int ORDER_STATUS_START = 1;
    public static final int ORDER_STATUS_DELIVERY = 2;
    public static final int ORDER_STATUS_COMPLETE = 3;

    //properties on this JavaBean
    private String toppings = null;
    private String address = null;
    private int orderID = 0;
    private int status = 0;
    private String email = null;
    private String creditCardNum = null;

    //getter and setter methods for the non-constant attributes
    ...
}
```

Create a Web service from a Java class

1. Select **CalculateTime.java** in the Services view.
2. Select **New --> Service built from** and select **Java**. Click **Next**.
3. Check the box next to the method name. Click **Next**.
4. Select **As a new file for each data type** radio button. Click **Finish**.

Similarly, you create a Web service that checks the customer's credit by evaluating the credit card number value entered in the form.

check.credit.CheckCredit.java with `checkCredit` method:

```
public boolean checkCredit(Order order)
{
    try {
        int num = Integer.parseInt(order.getCreditCardNum());
        if (num > 1000)
            return true;
        else
            return false;
    } catch (NumberFormatException e) {
        return false;
    }
}
```

Note: You can use the **service built from** wizard to generate Web services from other implementation types, such as EJB and other resource adapters.

Section 4. Create a business process

Introduction

The business process starts when an order is placed. It first checks whether the customer has a good credit history by calling the CheckCredit Web service. If the customer has bad credit, the process halts immediately and returns -1. Otherwise, it continues and invokes the CalculateTime Web service to calculate the time needed

for an order to be available.

Define the PlaceOrderPartner.wsdl file

This is the Web service interface for this process, PlaceOrderPartner.wsdl.

1. Create a service interface in Service Project PlaceOrderSP using **New --> Service Interface**.
2. Type *place.order* as the package, and `PlaceOrderPartner.wsdl` as the file name.
3. Keep the defaults and click **Finish**.
4. From the WSDL editor, perform the following tasks:
 1. Create a message called *placeOrderReqMsg*.
 2. Create a part for it called *order*, that has `Order.xsd` as the type.
 3. Create another message called *placeOrderResMsg*.
 4. Create a part for it called *result*, that has `xsd:int` as the type.
 5. Under `PlaceOrderPartner`, create an operation called *placeOrder*. It has an input and output that are called `placeOrderReqMsg` and `placeOrderResMsg`, respectively.

This file should now look like this in the WSDL editor:

Create a BPEL process

1. Select **New --> Business Process**.
2. Name the process *PlaceOrderProcess* in the *place.order* package. Click **Next**.
3. Select **Sequence-based BPEL** process. Click **Finish**.

There are four main areas in the BPEL editor: variables, correlation sets, partner links and process.

Define the partner links

1. Select **PartnerLink**.
2. From the Implementation tab, click **New**. Type the entries as shown in the New Partner Link Type window:
3. Click **OK**.
4. Change this role to be a Process Role by pressing the <--> button.

Create `CheckCreditPartnerLink` by clicking on the **+** sign next to Partner Links. A window opens and the information is entered as shown. Click **OK**. It has a partner role.

Repeat the steps to create `CalculateTimePartnerLink`.

Create variables in the process

From the editor, select **InputVariable**, and go to the Message page. Navigate to `placeOrderReqMsg` in the `PlaceOrderPartner.wsdl` file. Create the following variables and define their implementation:

- `OutputVariable` (`placeOrderResponse` message in `PlaceOrderPartner.wsdl`)
- `checkCreditReqMsg` (`checkCreditReqMsg` message in `CreditCheck.wsdl`)
- `checkCreditResMsg` (`checkCreditResMsg` message in `CreditCheck.wsdl`)
- `calculateTimeReqMsg` (`calculateTimeReqMsg` message in `CalculateTime.wsdl`)
- `calculateTimeResMsg` (`calculateTimeResMsg` message in `CalculateTime.wsdl`)

Create the content of the business process

To specify the implementation of an activity, select the node on the editor and go to the Implementation page in the details area. For example, after selecting the Receive activity, the details view shows the information about this node. The implementation page content for this node should look like this:

Reply

Partner Link: PlaceOrderPartnerLink
Port Type: PlaceOrderPartner
Operation: placeOrder
Response: OutputVariable

Assign

From Variable: InputVariable/placeOrderReqMsg/order
To Variable: checkCreditReqMsg/checkCreditRequest/order

CheckCredit

PartnerLink: CheckCreditPartnerLink
Operation: checkCredit
Request: checkCreditReqMsg
Response: checkCreditResMsg

Assign1

From Fixed value: xsd:int 1
To: OutputVariable/placeOrderResMsg/result

Assign2

From Variable: InputVariable/placeOrderReqMsg/order
To Variable: calculateTimeReqMsg/calculateTimeRequest/order

CalculateTimeNeeded

PartnerLink: CalculateTimePartnerLink
Request: calculateTimeReqMsg
Response: calculateTimeResMsg

Assign3

From Variable: calculateTimeResMsg/calculateTimeResponse/result
To Variable: OutputVariable/placeOrderResMsg/result

Link from CheckCredit to Assign1

Condition: Otherwise

Link from CheckCredit to Assign2

Condition: Expression

```
return getCheckCreditResMsg().getResult();
```

Deploy the business process

Since this business process is invoked through the Web, you need to deploy it with SOAP/HTTP bindings as well.

1. Specify the binding for the actual implementation.
2. Generate the proxy from the `PlaceOrderProcess_PlaceOrderPortType_SOAP.wsdl` file to the `PizzaWeb\JavaSource` folder. The idea is to create the proxy so that the Action class can call it.
3. Right-click the **PlaceOrderProcess_PlaceOrderPortType_SOAP.wsdl** file, and then select **Enterprise Services --> Generate Service Proxy**
4. Click **Next** and select the only operation in the navigation tree view.

After generating the proxy, there will be an error in the Web project. This error occurs because the process is not in the Web project's build path. Go to the J2EE Hierarchy view, expand Enterprise Applications and double-click **PizzaEAR**. Go to the module page, then add *PlaceOrderSP* to the Project Utility JAR files.

Invoke the Web service through a proxy

Return to the Web perspective. In `PlaceOrderAction.java`, which is in `PizzaWeb\JavaResources\com.ibm.pizzaweb.actions`, append the following snippet to the end of the try-catch block:

```
PlaceOrderPartnerProxy proxy = new PlaceOrderPartnerProxy();
Order order = new Order();
order.setAddress(address);

String creditCardNum = placeOrderForm.getCustomer().getCreditCardNum();
order.setCreditCardNum(creditCardNum);

order.setEmail(email);

if (size.equals(Order.SIZE_LARGE_STRING))
    order.setSize(Order.SIZE_LARGE);
else if (size.equals(Order.SIZE_MEDIUM_STRING))
    order.setSize(Order.SIZE_MEDIUM);
else
    order.setSize(Order.SIZE_SMALL);

order.setToppings(toppingsString);
timeNeeded = proxy.placeOrder(order);
request.setAttribute("timeNeeded", Integer.toString(timeNeeded));
```

Change the Action class for different forwarding JSP files

If the customer has bad credit, the process returns `-1`, so you don't want to tell the customer that an order has been placed successfully. Therefore, you create another Web page indicating that the order cannot be placed. In addition, the ActionMapping in the Struts framework needs to be changed.

1. Double-click **PizzaWeb\WebContent\WEB-INF\struts-config.xml**. In the Action mapping page, select the **Local Forwards** page that is shown at the top.
2. Select **/placeOrder** from Action Mappings.
3. Add *failSubmit* to Local Forwards, and define the forward attribute path to `./failSubmit.jsp`.
4. Edit the PlaceOrderAction class in the PizzaWeb project. Change from `forward = mapping.findForward("submit");` to:

```
// Write logic determining how the user should be forwarded.
if (timeNeeded > 0)
forward = mapping.findForward("submit");
else
    forward = mapping.findForward("failSubmit");
```

5. The previous snippet shows that if an order is placed successfully, the action mapping displays the submit.jsp file; otherwise, it displays failSubmit.jsp.
6. Create failSubmit.jsp. It can be one sentence to indicate the order cannot be placed.

```
                                Cannot order pizza.
<BR>Sorry,<%=request.getAttribute("firstname") %>.
<BR>You have bad credit.
```

7. Change confirm.jsp to display the time needed to prepare the order. Change `10` to `<%=request.getAttribute(timeNeeded)%>`.

Run the Web site

1. Switch to the Business Integration view.
2. Before restarting the server, add PlaceOrderSPEAR to the server.

3. From the server view, select the server. Right-click on the server name, and select **add and remove projects**.
 4. Click **Add All >>**. Then click **Finish**.
 5. Right-click **index.html** and select **Run on Server**.
-

Section 5. Summary

Summary

In this tutorial, you learned how to use WebSphere Studio Application Developer Integration Edition to develop a dynamic Web site with Struts. Using Java language as the implementation programming language, you learned how to generate Web services from a Java class. Finally, you learned how to compose a BPEL process that uses the Web services created in the previous step and how to invoke the process from the Struts Action class, which is called from a Web page.

Resources

Learn

- Learn more about the [Apache Struts Web Application Framework](#).
- Read the [Business Process Execution Language for Web Services specification \(BPEL\)](#).
- [Standards roadmap](#) -- understand the impact and importance of standards and specifications for the development of SOA and Web services.
- See the [Order Pizza Struts sample](#).
- Acquire more training through [presentations and labs](#) using WebSphere Studio Application Developer Integration Edition.
- [SOA and Web services](#) -- hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.

Get products and technologies

- Download the [IBM Business Process Execution Language for Web Services Java](#) from alphaWorks.
- [WebSphere Application Server](#) -- get a trial download.
- [WebSphere Studio Application Developer](#) -- get a trial download.

Discuss

- [developerWorks blogs](#) -- get involved in the developerWorks community.

About the author

Diana Lau

Diana is currently working in the SOA Advanced Technology Design Center in IBM Toronto Lab. Prior to that, she was working on the WebSphere Integration Developer product. Diana received a bachelor and master degree in Computer Science from the University of Toronto.