IBM WebSphere Application Server v5.0 Workshop – LAB EXERCISE

A Login Servlet and an Access Filter

What this exercise is about

In this lab you will create a servlet to handle login requests to the Go-ForIt site. You will also create a Servlet 2.3 filter to protect resources that are accessed after a user logs in.

What You Should Be Able To Do

At the completion of this lab exercise, you should be able to create and test servlets and servlet filters using WebSphere Studio Application Developer v5.0

Introduction

In this exercise you will create and test a servlet to handle user logins and then you'll create and test a servlet filter that will protect resources that can only be accessed after a user has successfully logged in.

You will be creating the Login servlet which will be processing a login based on the user information that has already been provided in the GOFORIT database.The login form (index.jsp) has also already been provided to you by the web designers and contains an HTML form with input fields for the username and password and a Submit button. Pressing the **Submit** button will cause the browser to invoke the LoginServlet on the server. The Login servlet will verify the ID and password by checking the entry in the USER table of the GOFORIT database. Access to the database is via JDBC.

If the ID and password are valid the user is sent onto the CustomerMainMenuServlet which presents the main menu page. If there is an error, you will return to the Login page with an error message displayed.

The LoginServlet will also store a data bean with user information retrieved from the database during the login request in Session scope (i.e. HttpSession) to serve as an indicator that a user has logged in.

After completing the servlet, you'll create a servlet filter to protect resources that require the user to be logged in to access (e.g. the CustomerMainMenuServlet). The filter will retrieve the HttpSession instance and check for a valid session containing the user data bean that is added during the login process and redirect to the login page if the session has expired. If a valid session is found, the filter will continue on to the resource being accessed.

To support the filter, all protected JSP pages in the application are under the WEB-INF folder of the web application (preventing direct browser access) and all protected servlets have a URI mapping that starts with **/protected**. The filter is configured to intercept all requests to URIs that begin with **/protected**. Protected JSP pages are accessed via servlets.

Exercise Instructions

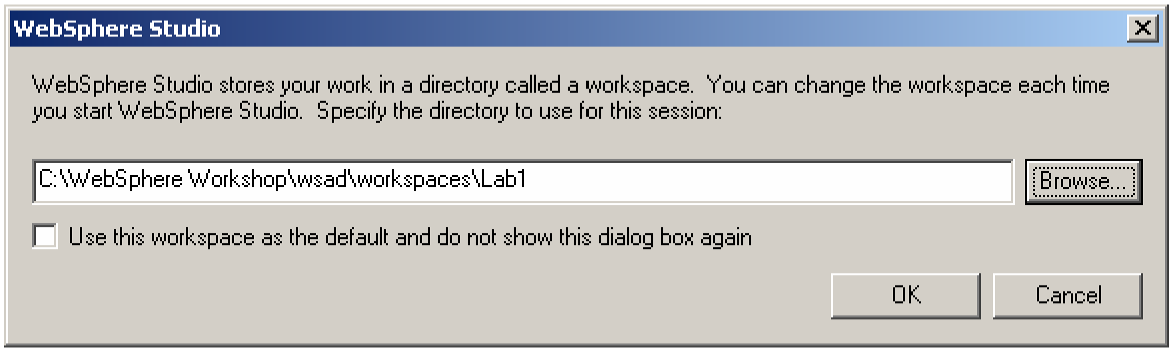
Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__1. Start WebSphere Studio Application Developer

__a. From the Windows Start Menu select **Start>Programs>IBM WebSphere Studio>Application Developer**

__b. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab1**



WebSphere Studio stores your work in a directory called a workspace. You can change the workspace each time you start WebSphere Studio. Specify the directory to use for this session:

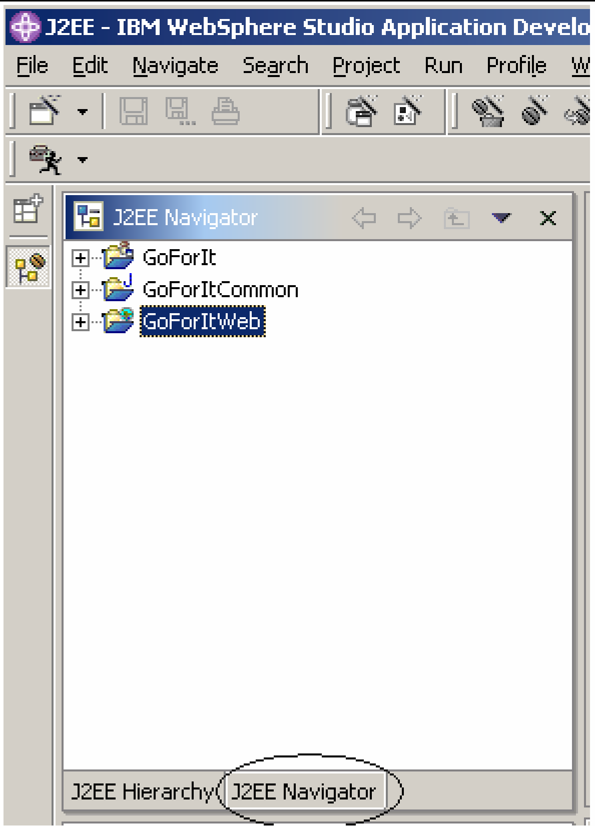
C:\WebSphere Workshop\wsad\workspaces\Lab1

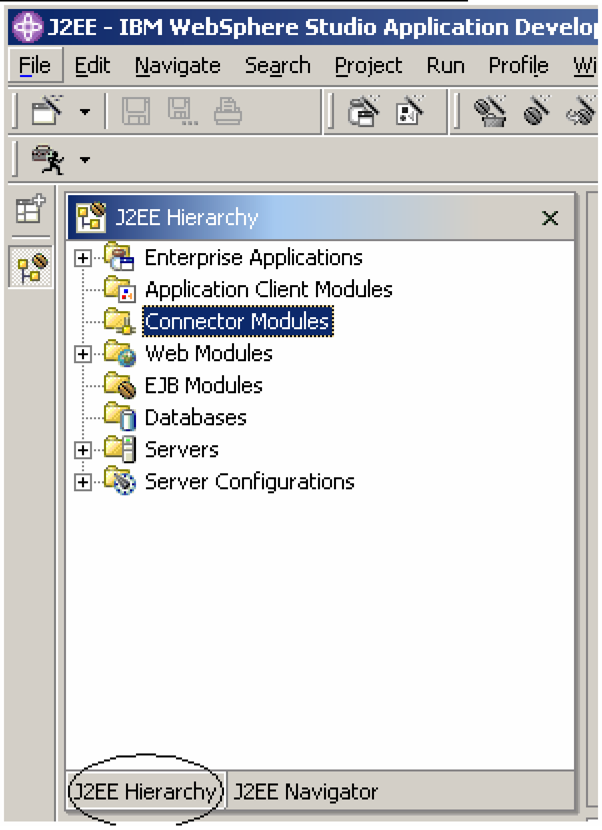
☐ Use this workspace as the default and do not show this dialog box again

__c. Click **OK**

__2. As mentioned in the WebSphere Studio Application Developer introductory lecture, Application Developer contains several perspectives from which you can work with. For the labs we will be working almost exclusively in the J2EE Perspective.

__a. There are two views that we will work with. If you look at the pane in the top left of your screen see the two views as shown in the following diagrams


J2EE - IBM WebSphere Studio Application Develo
File Edit Navigate Search Project Run Profile W
J2EE Navigator
GoForIt
GoForItCommon
GoForItWeb
J2EE Hierarchy
J2EE Navigator



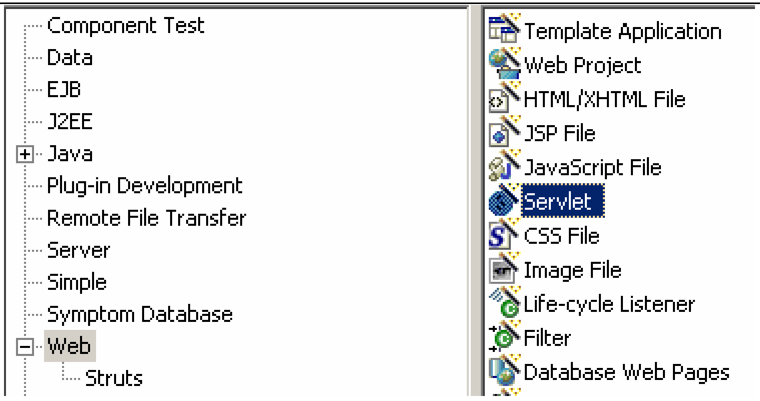
__b. Make sure you can switch between the two views (J2EE Navigator and J2EE Hierarchy) by clicking on the appropriate tab on the bottom. During the lab instructions we will direct you to use the appropriate view.

Part One: Create the LoginServlet

__1. Start by using the Servlet Wizard to create the LoginServlet

__a. Select **File->New->Other** from the main menu

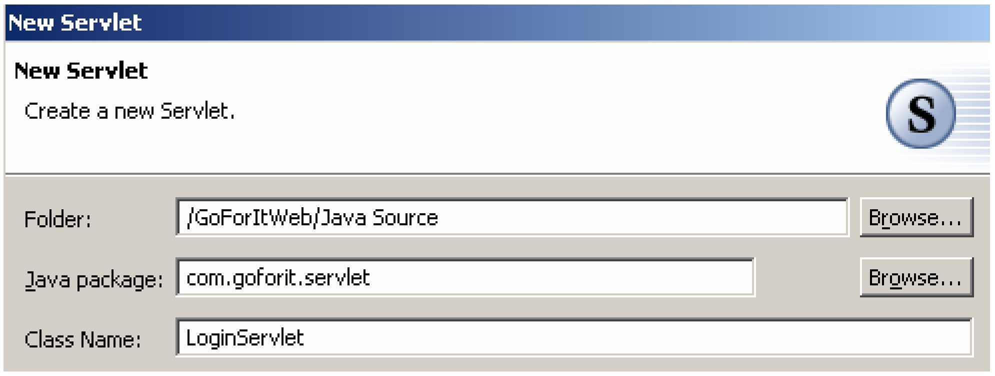
__b. Select **Web** from the pane on the left and **Servlet** from the pane on the right and click **Next**



__c. On the next screen use the **Browse** button at the right of the **Folder** field to select **/GoForItWeb/Java Source**

__d. Use the Browse button at the right of the **Java package** field to select **com.goforit.servlet** as the Java package

__e. Enter **LoginServlet** as the **Class Name**


New Servlet

New Servlet

Create a new Servlet.

Folder: /GoForItWeb/Java Source

Java package: com.goforit.servlet

Class Name: LoginServlet

__f. Click **Finish**

__2. The wizard brings up LoginServlet.java in the Java Editor. We'll route all calls to doGet() and doPost() through a method called performTask() for convenience

__a. Add the following method template to LoginServlet.java

```
public void performTask(
   HttpServletRequest req,
   HttpServletResponse res)
   throws ServletException, IOException {

}
```

__3. Add calls to performTask() to both doGet() and doPost().

__a. Add the following line of code to both methods

```
performTask(req, resp);
```

__4. Complete the performTask() method.

__a. Add the following to the body of the performTask() (Note: To save time, you can copy the code from C:\WebSphere Workshop\snippets\lab1\performTask.txt)

```
try {

  // Look for userid and password in
  // request
  String userid = req.getParameter("userid");
  String password = req.getParameter("password");

  // Send to back to login page if missing username of password
  if ((userid == null) || (password == null)) {
      res.sendRedirect(req.getContextPath() + "/index.jsp");
      return;
   }

 // Login in the user
 // Get the Business Delegate implementation, the factory pattern is
 // used so this code can be used with different backends (e.g. JDBC
 // or EJBs)


 CustomerBusinessDelegate businessDelegate =
                   CustomerBusinessDelegateFactory.getImpl();


 // Login the user

 UserDataBean user = businessDelegate.login(userid, password);
 HttpSession session = req.getSession(true);
 session.setAttribute("user", user);
 res.sendRedirect(req.getContextPath() +
               "/protected/CustomerMainMenuServlet");
 return;

} catch (BusinessDelegateException e) {
 // In case of error go back to login page with error message
  req.setAttribute("errorMsg", new ErrorView(e.getMessage()));
  getServletContext().getRequestDispatcher("/index.jsp").forward(
                  req,
                  res);
  return;

}
```

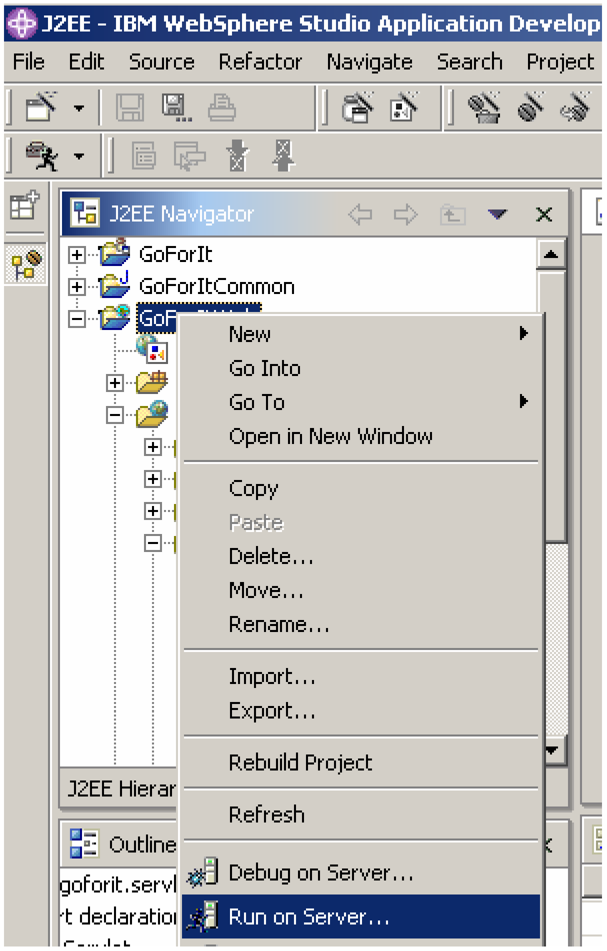
__b. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__c. Format the code you just added by entering **Ctrl+Shift+F**

__d. Save the file by entering **Ctrl+S**

Part Two: Test the LoginServlet

__1. Start the WebSphere 5.0 Test Environment in Application Developer __a. From the **J2EE Navigator** view, right click on the **GoForitWeb** project and select **Run on server**. This will start the Test Environment and bring up the GoForIt application to its configured start page (**index.jsp**) on your browser



__2. After the server starts, the browser will come up. Test the LoginServlet by logging in

__a. Enter **joeg** as the userid and **password** as the password and click on **Login** . Verify that the main menu page comes up.

__b. Click on the **Logout** icon (at the top right) and login again as **joeg** using a password of **foo** (or anything other than **password**). Verify that the login page comes up again with an error message.

__3. Verify that you can access the CustomerMainMenuServlet without logging in. This will be fixed in the next part of the lab.

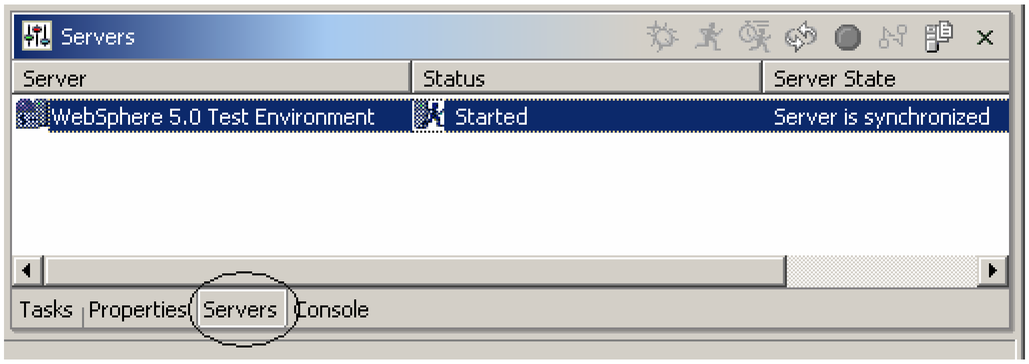
__a. Shutdown the browser.

__b. Start the browser again and open the URL http://localhost:9080/GoForItWeb/protected/CustomerMainMenuServlet . Verify that the Main Menu page comes up.

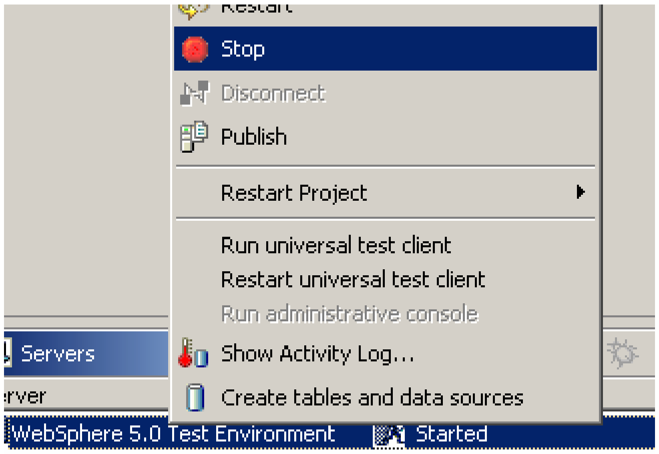
Part Three: Create an access filter

__1. Stop the WebSphere 5.0 Test Environment

__a. Click on the **Servers** tab in the Tasks pane at the bottom right



__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop**

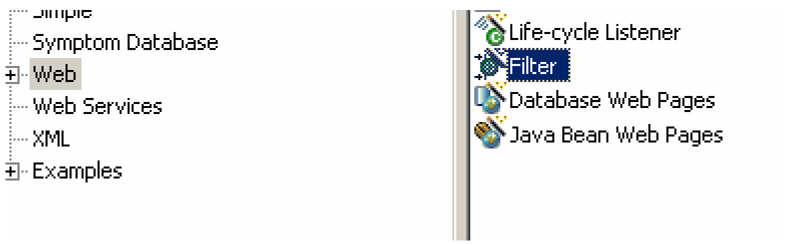

Restart
Stop
Disconnect
Publish
Restart Project
Run universal test client
Restart universal test client
Run administrative console
Show Activity Log...
Create tables and data sources
Servers
rver
WebSphere 5.0 Test Environment
Started

__2. Create a servlet filter using the wizard

__a. Select **File->New->Other** from the main menu

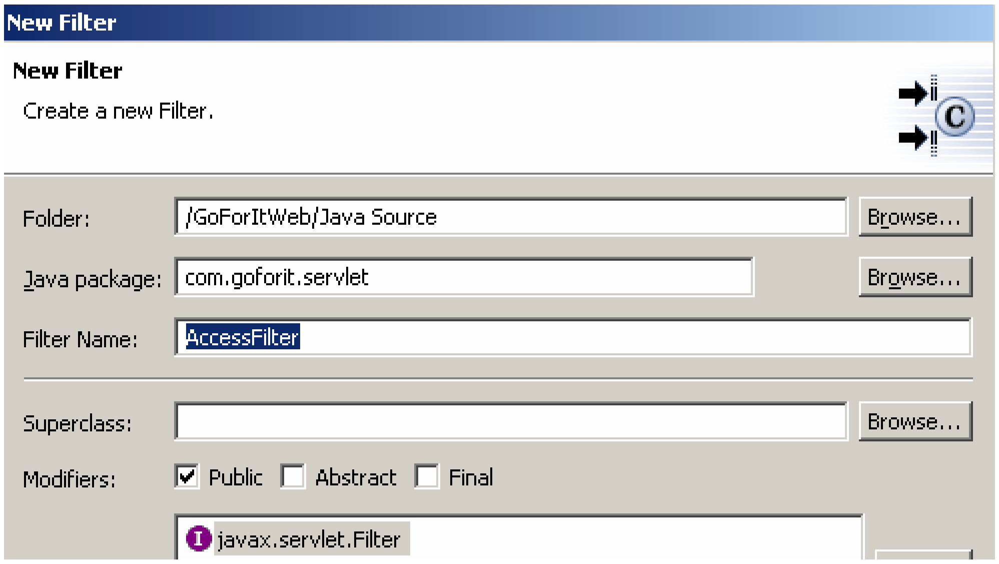
__b. Select **Web** from the pane on the left and **Filter** from the pane on the right and click **Next**

__c.



__c. On the next screen use the **Browse** button at the right of the **Folder** field to select **/GoForItWeb/Java Source** __d. Use the Browse button at the right of the **Java package** field to select **com.goforit.servlet** as the Java package

__e. Enter **AccessFilter** as the **Filter Name**


New Filter

New Filter

Create a new Filter.

Folder: /GoForItWeb/Java Source

Java package: com.goforit.servlet

Filter Name: AccessFilter

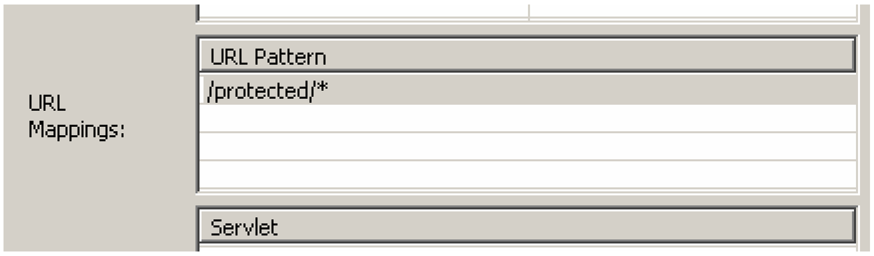
Superclass:

Modifiers: [✓ Public] Abstract Final

javax.servlet.Filter

__f. Click **Next**

__g. Under **URL Mappings** highlight the URL Pattern **/AccessFilter** and change it to **/protected/***


URL Mappings:

URL Pattern
/protected/*

__h. Click on **Finish**

__3. The wizard brings up AccessFilter.java in the Java editor. Save the FilterConfig instance passed to the init() method

__a. Add the following instance variable to AccessFilter.java

```
FilterConfig filterConfig = null;
```

__b. Add the following to the init() method

```
filterConfig = config;
```

__4. Complete the doFilter() method

__a. Replace the code in the doFilter() method with the following (Note: To save time, you can copy the code from C:\WebSphere Workshop\snippets\lab1\doFilter.txt)

```
// Check for existing  session
HttpServletRequest request = (HttpServletRequest) req;
HttpServletResponse response = (HttpServletResponse) resp;
UserDataBean user = null;
HttpSession session = request.getSession(false);
if (session != null) {
      user = (UserDataBean) session.getAttribute("user");
}

// Continue on to requested page if session exists
if (user != null) {
      chain.doFilter(req, resp);

}
 // Back to login if session doesn't exist
else {
      filterConfig.getServletContext().getRequestDispatcher(
            "/index.jsp").forward(
            request,
            response);

}
```

__b. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__c. Format the code you just added by entering **Ctrl+Shift+F**

__d. Save the file by entering **Ctrl+S**

Part Four: Test the AccessFilter

__1. Start the WebSphere 5.0 Test Environment in Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Start**. Wait for the message in the console that says

Server server1 open for e-business before continuing

__2. Test the filter by accessing a protected resource

__a. Start your browser and go to the URL http://localhost:9080/GoForItWeb/protected/CustomerMainMenuServlet . Verify that the login page comes up.

__b. Login with userid **joeg** and a password of **password**. Verify that the Main menu page comes up.

__3. Stop the server and shutdown Application developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop**

__c. Shutdown Application Developer (**File->Exit**)

What you did in this exercise

You created and tested a servlet to handle the login function of the Go-ForIt web site. You also created and tested a servlet filter to ensure that only logged in users can answer protected areas of the site.

IBM WebSphere Application Server v5.0 Workshop – LAB EXERCISE

Struts and JSTL with Application Developer

What this exercise is about

In this lab you will work with the Struts Application Development Environment in Application Developer and also work with JSTL to handle complex rendering of application data in a JSP page. You'll redo the login function of the previous lab using Struts.

What You Should Be Able To Do

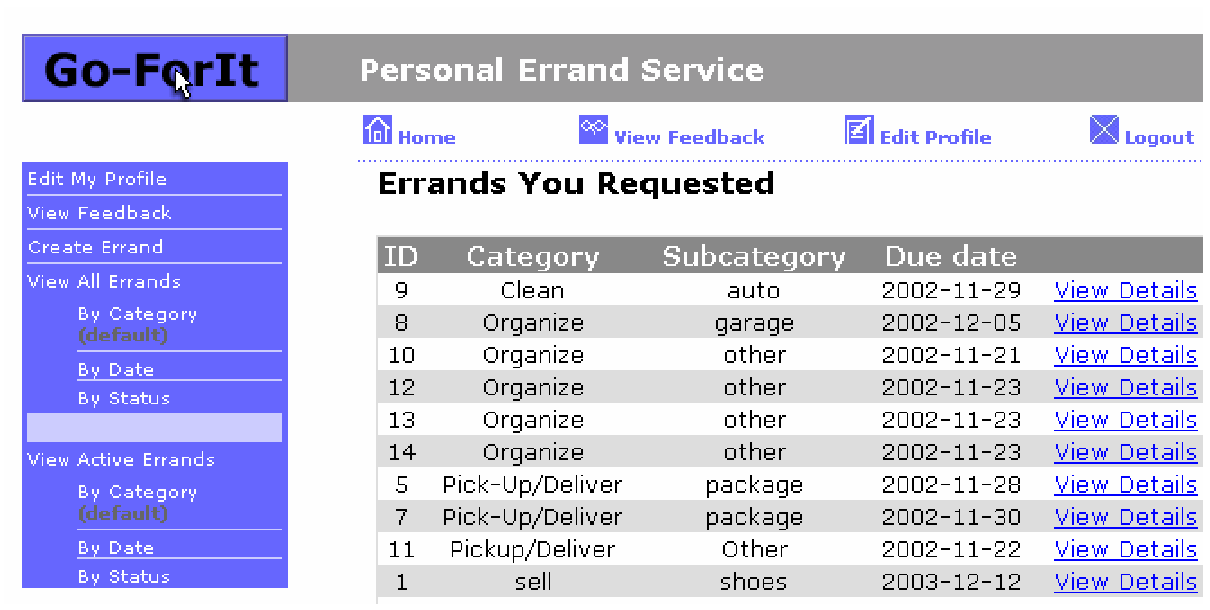
At the completion of this lab exercise, you should be able to create and test Struts components using WebSphere Studio Application Developer v5.0 and to use JSTL core tags in your JSP pages.

Introduction

In this exercise you will redo the login function using Struts and use JSTL to display a list of a users errand requests using JSTL.

You'll create a Struts Action to process user logins (as the LoginServlet did in the previous lab) and create a Struts Form bean to encapsulate and validate the input entered by the user on the login page. This will be done using the Struts based visual editor in Application Developer.

In the second part of the lab, you'll use JSTL to dynamically generate a list of errands as a summary and a series of links to the details of each errand in the list as shown below.



Errands You Requested

ID	Category	Subcategory	Due date	
9	Clean	auto	2002-11-29	View Details
8	Organize	garage	2002-12-05	View Details
10	Organize	other	2002-11-21	View Details
12	Organize	other	2002-11-23	View Details
13	Organize	other	2002-11-23	View Details
14	Organize	other	2002-11-23	View Details
5	Pick-Up/Deliver	package	2002-11-28	View Details
7	Pick-Up/Deliver	package	2002-11-30	View Details
11	Pickup/Deliver	Other	2002-11-22	View Details
1	sell	shoes	2003-12-12	View Details

Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__1. Start WebSphere Studio Application Developer

__a. From the Windows Start Menu select **Start>Programs-IBM WebSphere Studio>Application Developer**

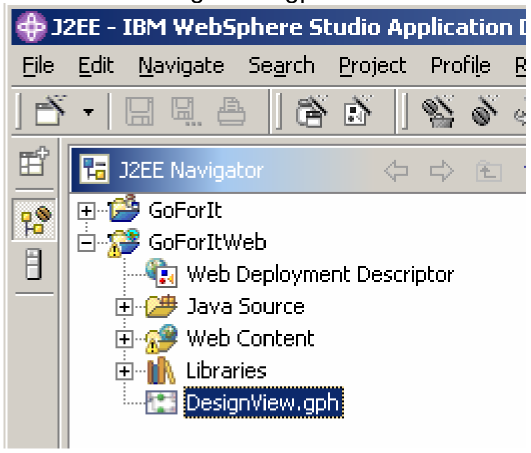
__b. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab2**

__c. Click **OK**

Part One: Create the Struts Components

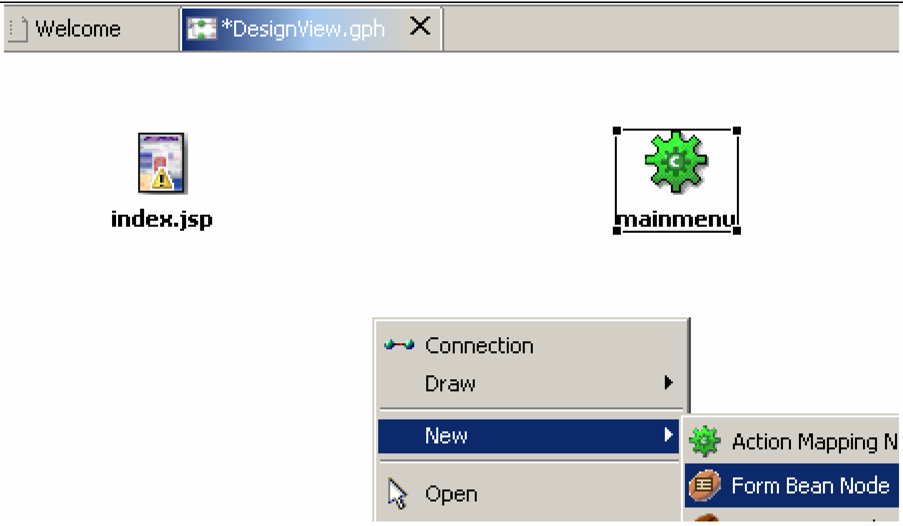
__1. Start the Struts Visual Development Environment

__a. From the J2EE Navigator view, expand the **GoForItWeb** project and double click on DesignView.gph

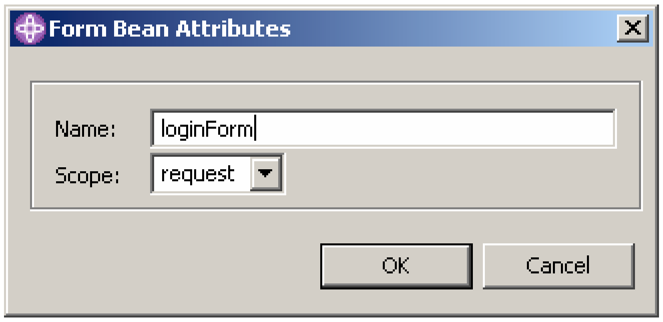

J2EE - IBM WebSphere Studio Application
File Edit Navigate Search Project Profile
J2EE Navigator
GoForIt
GoForItWeb
Web Deployment Descriptor
Java Source
Web Content
Libraries
DesignView.gph

__2. Start adding the required Struts components. The login page and the mainmenu action (which brings up the mainmenu) have already been added for you.

__a. Right click in the Visual editor and select **New>Form Bean Node**



__b. Click anywhere in the editor's window to add the Form Bean and enter **loginForm** as the name when prompted

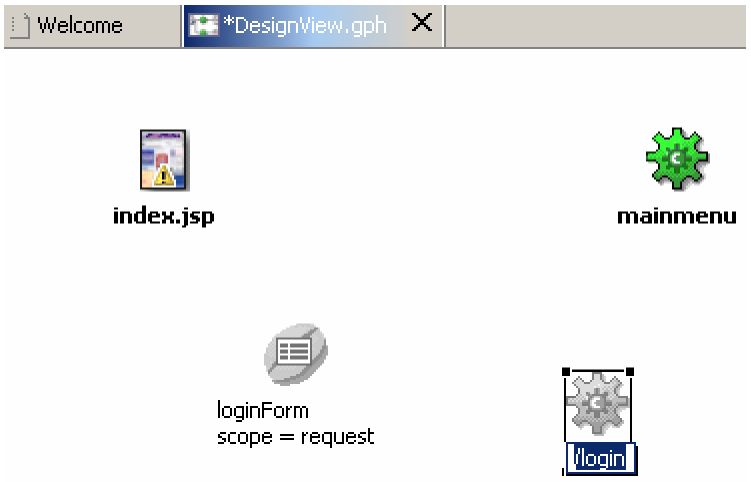


Name: loginForm
Scope: request

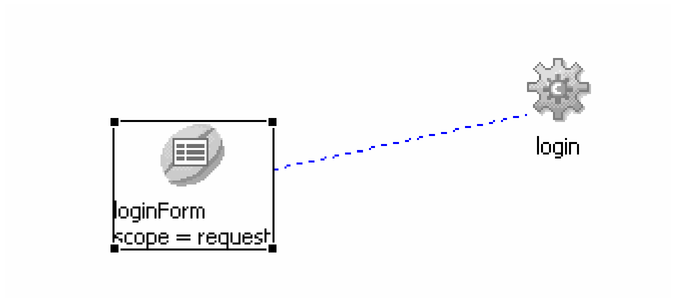
__**c.** Click **OK**

__d. Right click in the editor and select **New>Action Mapping Node**

__e. Click anywhere in the editor's window to add the Action Mapping Node and enter **/login** as the name

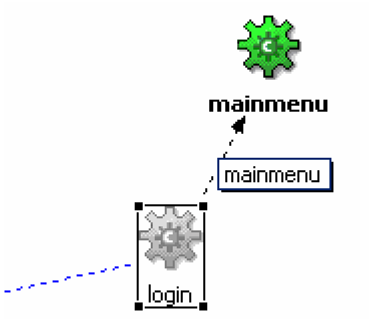


__f. Click on the **loginForm** node in the diagram, right click and select **Connection** __g. Click on the **login** node to complete the connection



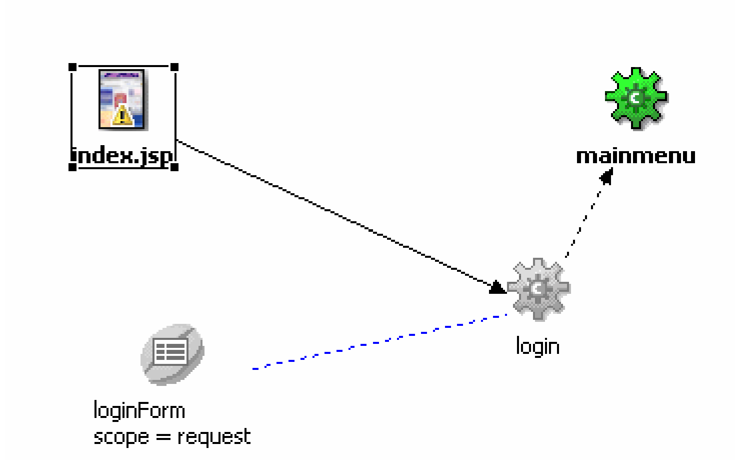
__h. Select the **login** node, right click and select Connection

__i. Click on the **mainmenu** node and type in **mainmenu** as the forward name to complete the connection



__j. Select the **index.jsp** node, right click and select **Connection**

__k. Click on the **login** node to complete connection

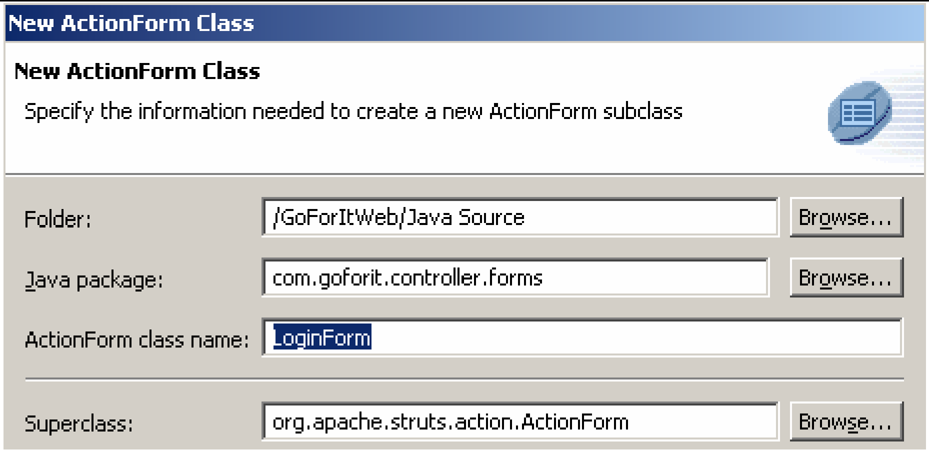


__l. Type **Ctrl+S** to save the diagram

__3. From the diagram, we can launch all the wizards we need to create the added components

__a. Double click on the **loginForm** node to bring up the Action Form wizard

__b. Use the Browse button at the right of the **Java package** field to select **com.goforit.controller.forms** as the Java package and type in **LoginForm** as the **Action form class name**


New ActionForm Class

New ActionForm Class

Specify the information needed to create a new ActionForm subclass

Folder: /GoForItWeb/Java Source

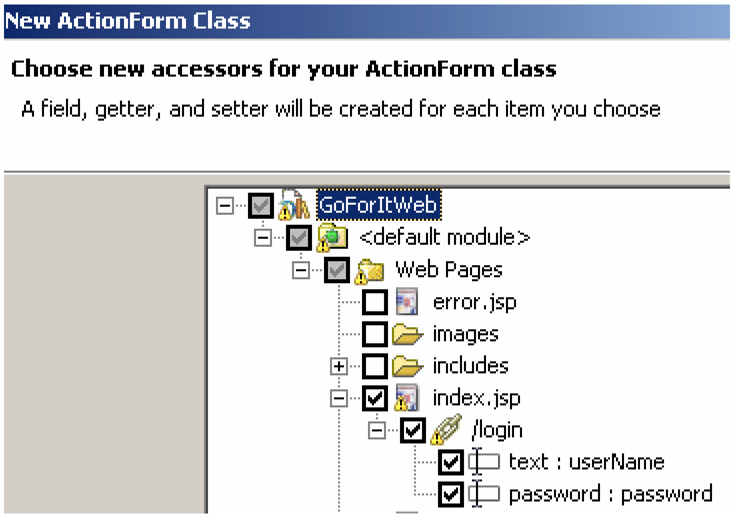
Java package: com.goforit.controller.forms

ActionForm class name: LoginForm

Superclass: org.apache.struts.action.ActionForm

__c. Click **Next**

__d. Expand **GoForItWeb**, **<default module>**, **Web Pages** and select **index.jsp** as shown below to select the fields that need to be added to the form bean


New ActionForm Class

Choose new accessors for your ActionForm class

A field, getter, and setter will be created for each item you choose

__e. Click **Next**

__f. The next page shows the accessors to be added to the form bean. Click **Next**

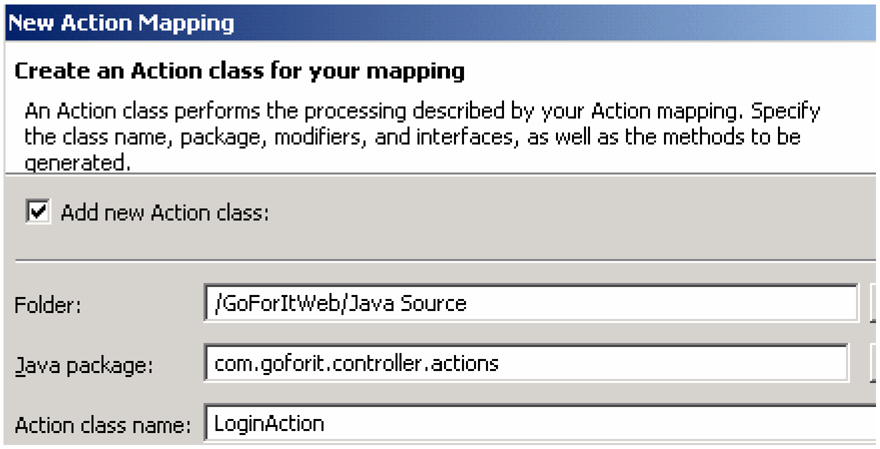
__g. Click **Finish.** This should bring up the new form bean in the Java Editor.

__h. Return to **DesignView.gph** and double click on the **login** node to bring up the Action wizard

__i. Under the **Forwards** section select the existing **mainmenu** forward and click on **Remove** (we'll use a globally defined forward for this instead)

__j. Click **Next**

__k. Double click on the **login** node to bring up the Action Mapping wizard __l. Use the Browse button at the right of the **Java package** field to select **com.goforit.controller.actions** as the Java package and type in **LoginAction** as the **Action class name**


New Action Mapping

Create an Action class for your mapping

An Action class performs the processing described by your Action mapping. Specify the class name, package, modifiers, and interfaces, as well as the methods to be generated.

☑ Add new Action class:

Folder: /GoForItWeb/Java Source

Java package: com.goforit.controller.actions

Action class name: LoginAction

__m. Click **Finish**. This brings up the Action class in the Java Editor.

__4. Add the code to the classes that you just generated

__a. Replace the generated contents of the execute() method in LoginAction.java with the following (Note: You can copy the code from C:\WebSphere Workshop\snippets\lab2\loginAction.txt)

```
LoginForm loginForm = (LoginForm) form;

UserDataBean user = null;
ActionErrors errors = new ActionErrors();

/*
 * Go to  database  to validate user
 */
try {
      // Get the Business Delegate implementation

      CustomerBusinessDelegate businessDelegate =
            CustomerBusinessDelegateFactory.getImpl();

      /*
       * Add the errand
       */
      user =
            businessDelegate.login(
                  loginForm.getUserName(),
                  loginForm.getPassword());

} catch (InvalidUserIdException e) {
      /*
      * Login exceptions go back to registration page w/error
      */
      errors.add(
            ActionErrors.GLOBAL_ERROR,
            new ActionError("error.invalid.username"));
      saveErrors(request, errors);
      return (new ActionForward(mapping.getInput()));

} catch (InvalidPasswordException e1) {
```

```
            /*
             * Login exceptions go back to registration page
            *  w/error
             */
                  errors.add(
                        ActionErrors.GLOBAL_ERROR,
                        new ActionError("error.invalid.password"));
                  saveErrors(request, errors);
                  return (new ActionForward(mapping.getInput()));

            }
            // Add user info to session state and go to main menu
            HttpSession session = request.getSession(false);
            if (session != null) {
                  session.invalidate();
            }
            session = request.getSession(true);
            session.setAttribute("user", user);

            return (mapping.findForward("mainmenu"));

      }
```

__b. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__c. Format the code you just added by entering **Ctrl+Shift+F**

__d. Save the file by entering **Ctrl+S**

__e. Go to **LoginForm.java** and replace the contents of the **validate()** method with the following (Note: You can copy the code from C:\WebSphere Workshop\snippets\lab2\loginForm.txt)

```
ActionErrors errors = new ActionErrors();
// Validate the fields in your form, adding
// adding each error to this.errors as found, e.g.

if ((userName == null) || (userName.length() < 1)) {
   errors.add("username",
                  new ActionError("error.username.required"));
}

if ((password == null) || (password.length() < 1)) {
   errors.add("password",
                  new ActionError("error.password.required"));
}

return errors;
```

__f. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__g. Format the code you just added by entering **Ctrl+Shift+F**

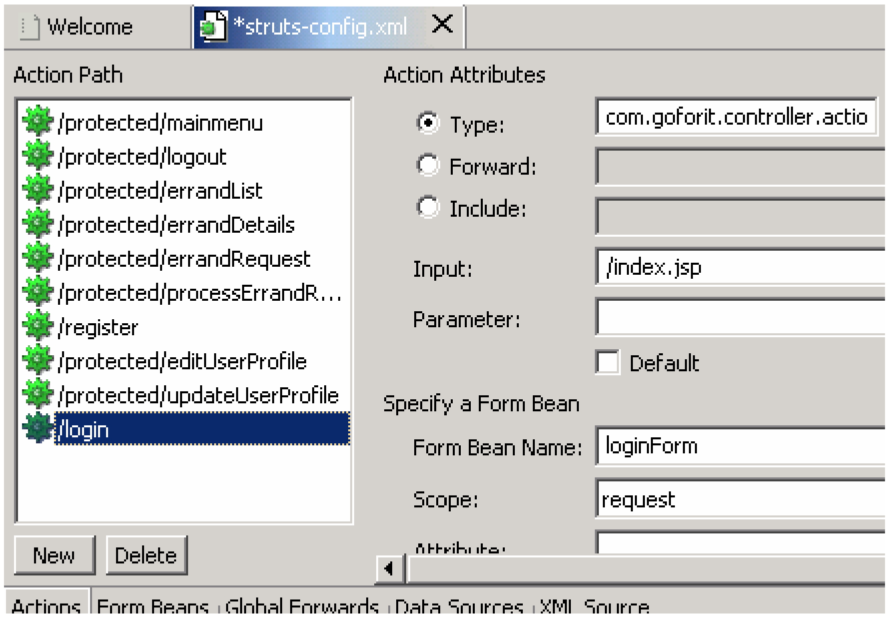
__h. Save the file by entering **Ctrl+S**

__5. Edit the Struts configuration file to complete the configuration

__a. From the J2EE Navigator view, expand **GoForItWeb, Web Content, WEB-INF** and double click on **struts-config.xml** to bring up the Struts Configuration editor.

__b. Select the **Actions** tab at the bottom and select the **/login** action

__c. Enter **/index.jsp** as the **Input**



Type: com.goforit.controller.actio

Input: /index.jsp

Specify a Form Bean

Form Bean Name: loginForm

Scope: request

__d. Save the file by entering **Ctrl+S**

Part Two: Test the Struts Components

__1. Start the WebSphere 5.0 Test Environment in Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Start**. Wait for the message in the console that says

Server server1 open for e-business before continuing

__2. Test the Struts components by logging in

__a. Start your browser and go to the URL **http://localhost:9080/GoForItWeb/index.jsp**.

__b. Login with userid **joeg** and a password of **password**. Verify that the Main menu page comes up.

__c. Click on the **Logout** icon and try logging in again with the wrong password. Verify that the login page is reloaded with an error message.

__d. Logout, and then login again without providing a password and verify that the validation logic in the form bean works as expected

__3. Shutdown the server

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop**

Part Three: Adding the JSTL tags to the errand list request result page

__1. Add the JSTL to the JSP the displays the results of an errand list request

__a. From the J2EE Navigator view, expand **GoForItWeb, Web Content, WEB-INF, pages** , right click on **errand_list.jsp** and select **Open With > Source Editor**

__b. In the editor scroll down to the comment line that says **<!-- TO DO Add formatted list -->** and add the following (Note: To save time, you can copy the code from C:\WebSphere Workshop\snippets\lab2\jstl.txt)

```
<c:choose>

    <%-- No errands in errandList bean or null --%>
   <c:when test="${empty errands}">
    <P><B> You haven't requested any errands !</B>
   </c:when>

    <%-- There are errands, format as a table --%>
   <c:otherwise>
    <TABLE class="errandListTable" cellspacing="0">
    <TR class="errandListTableHeader">
    <TH>ID</TH>
    <TH>Category</TH>
    <TH>Subcategory</TH>
    <TH>Due date</TH>
    <TH>&nbsp;</TH>
      </TR>

    <c:forEach items="${errands}" var="eachErrand" varStatus="stat">

       <%-- Alternate background color of each row in the table  --%>
       <c:set var="cssClassName" value="errandListTableRow" />
       <c:if test="${stat.index % 2 == 1}">
        <c:set var="cssClassName" value="errandListTableRow_AlternatingColor" />
       </c:if>

       <TR class="<c:out value="${cssClassName}" />">
       <TD align="center"><c:out value="${eachErrand.id}"/></TD>
       <TD align="center"><c:out value="${eachErrand.category}"/></TD>
       <TD align="center"><c:out value="${eachErrand.subcategory}"/></TD>
       <TD align="center"><c:out value="${eachErrand.dueDate}"/></TD>
       <TD align="center">

         <%-- URL to details of each errand --%>
```

```
             <c:url var="details" value="/protected/errandDetails.do">
               <c:param name="id" value="${eachErrand.id}"/>
             </c:url>
            <a href='<c:out value="${details}"/>'>View Details</a>
           </TD>
          </TR>
       </c:forEach>

       </TABLE>

       </c:otherwise>

   </c:choose>
```

__c. Read through the added code and make sure you understand what it's doing

__d. Save the file by entering **Ctrl+S**

Part Four: Test the JSTL tags

__1. Start the WebSphere 5.0 Test Environment in Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Start**. Wait for the message in the console that says
Server server1 open for e-business before continuing

__2. Test the Struts components by looking at a list of errands

__a. Start your browser and go to the URL **http://localhost:9080/GoForItWeb/index.jsp**.

__b. Login with userid **joeg** and a password of **password**. Verify that the Main menu page comes up.

__c. Click on the **View all errands** link and verify that a list of errands is displayed along with a link to view the details of each errand

__d. Click on **View Details** for one of the errands and verify that details are displayed.

__3. Shutdown the server and Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop** __c. Shutdown Application Developer (**File->Exit**)

What you did in this exercise

In this exercise you created and tested Struts components using Application Developer and learned to use JSTL core tags in your JSP pages.

IBM WebSphere Application Server v5.0 Workshop – LAB EXERCISE

Deploying Web Applications to WebSphere 5.0

What this exercise is about

In this exercise, you will deploy the web application that you have thus far developed and tested within Application Developer. In the process you will work with the web based WebSphere 5.0 Admininstration Console.

What You Should Be Able To Do

You should be able to export from Application Developer , create a JDBC provider, and DataSource with the WebSphere Admin Console and deploy an EAR file to WebSphere.

Introduction

In this exercise you will export the Go-ForIt Enterprise Application from Application Developer as an EAR file and then start the WebSphere Admin Server and launch the Admin Console. Using the Admin Console, you will create a new JDBC provider and the DataSource that is used by the Go-ForIt application to access the database . The exported EAR file will then be installed (deployed) to the Application server. Finally you'll start the application server and test the deployed application.

Exercise Instructions

Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__1. Start WebSphere Studio Application Developer

__a. From the Windows Start Menu select **Start>Programs-IBM WebSphere Studio>Application Developer**

__b. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab3**

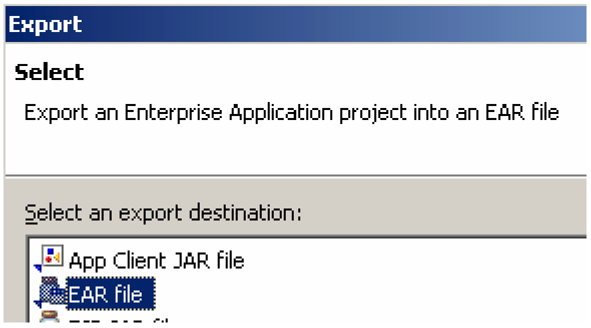
__c. Click **OK**

Part One: Exporting from Application Developer

__1. Run the EAR export wizard to export the Go-ForIt application

__a. Select **File->Export** from the main menu

__b. Select **EAR file** as the export destination


Export

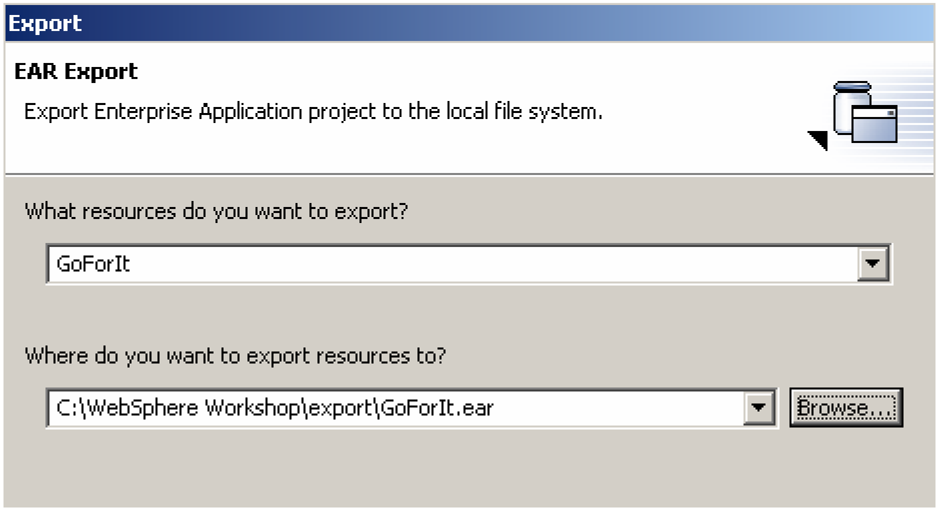
Select

Export an Enterprise Application project into an EAR file

Select an export destination:

__c. Click **Next**

__d. Select **GoForIt** as the resource to export and save it as **GoForIt.ear** in the **C:\WebSphere Workshop\export** folder.



Export Enterprise Application project to the local file system.

What resources do you want to export? GoForIt

Where do you want to export resources to? C:\WebSphere Workshop\export\GoForIt.ear

__e. Click **Finish**

__2. Exit from Application Developer

__a. **File->Exit** from the main menu

Part Two: Configuring WebSphere and Deploying the Enterprise Application

__1. Start the WebSphere Application Server

__a. Open a Command Prompt and change to the **C:\Program Files\WebSphere\AppServer\bin** directory

__b. Enter **startserver server1** at the command prompt

__c. The server will be fully started when the process ID is listed as shown below (Note: your process ID will most likely be different from the one shown below)

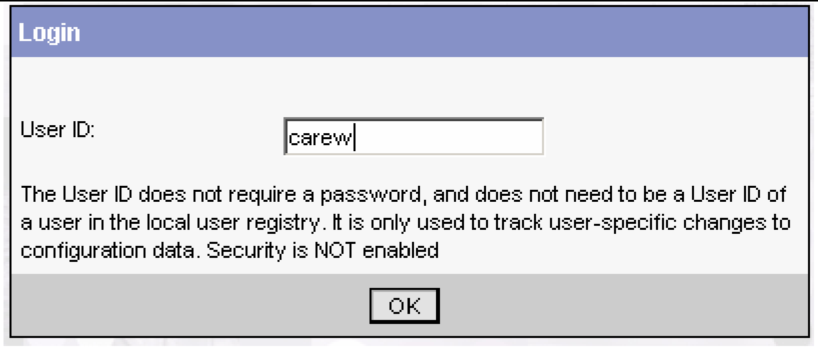
```
C:\Program Files\WebSphere\AppServer\bin>startserver server1
ADMU0116I: Tool information is being logged in file C:\Program
           Files\WebSphere\AppServer\logs\server1\startServer.log
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 1772
```

__2. The GoForIt application accesses the database using JDBC via a DataSource. You'll have to configure the DataSource in order for the application to run correctly.

__a. Start the WebSphere Admin Console by selecting the following from the Windows Start Menu

Start>Programs>IBM WebSphere Application Server v5.0>Administrative Console

__b. The console will open in your browser with a prompt for a user ID. In Enterprise Edition the IDs can be defined with different levels of authority , for the base version this feature is not supported. Enter any ID and click on **OK**.

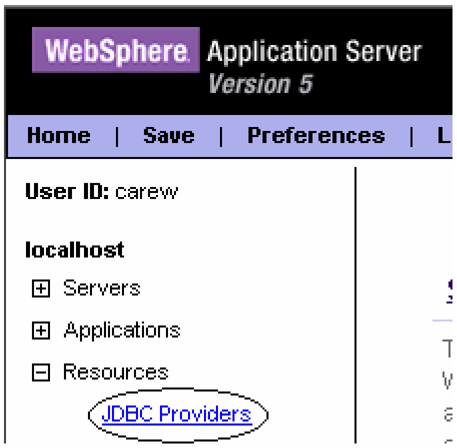


Login

User ID: carew

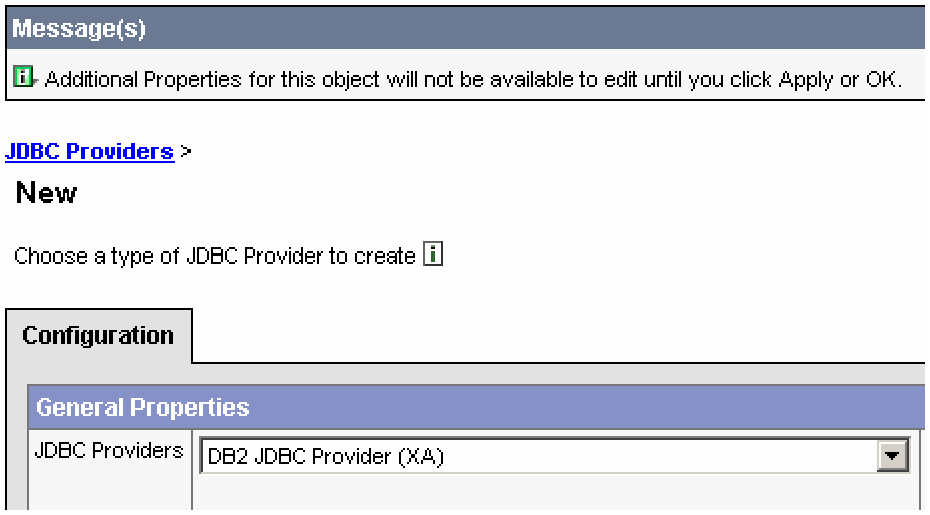
The User ID does not require a password, and does not need to be a User ID of a user in the local user registry. It is only used to track user-specific changes to configuration data. Security is NOT enabled

__c. Expand the Resources tree on the left and click **JDBC Providers** to begin the configuration of the DataSource



__d. In the Work area frame on the right, click on **New**

__e. Select **DB2 JDBC Provider (XA)** from the pull down list and click on **OK**



Message(s)

Additional Properties for this object will not be available to edit until you click Apply or OK.

JDBC Providers >

New

Choose a type of JDBC Provider to create

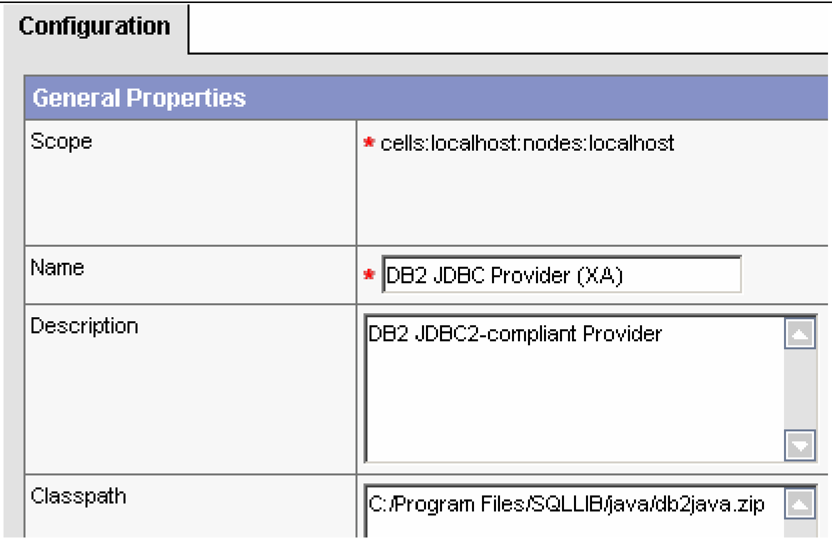
Configuration

General Properties

JDBC Providers: DB2 JDBC Provider (XA)

__f. The configuration properties for the JDBC Provider will be displayed. They are all correct except for the classpath. Enter the following for the classpath (note that the use of forward slashes is **not** a typo)

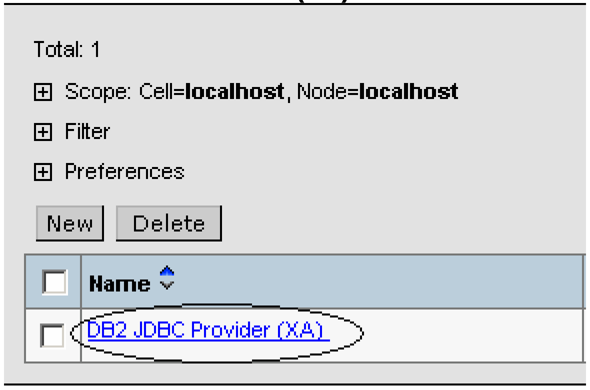
C:/Program Files/SQLLIB/java/db2java.zip



Scope	* cells:localhost:nodes:localhost
Name	* DB2 JDBC Provider (XA)
Description	DB2 JDBC2-compliant Provider
Classpath	C:/Program Files/SQLLIB/java/db2java.zip

__g. Click **OK**

__h. With the JDBC Provider created you can now create the DataSource. Click **DB2 JDBC Provider (XA)**



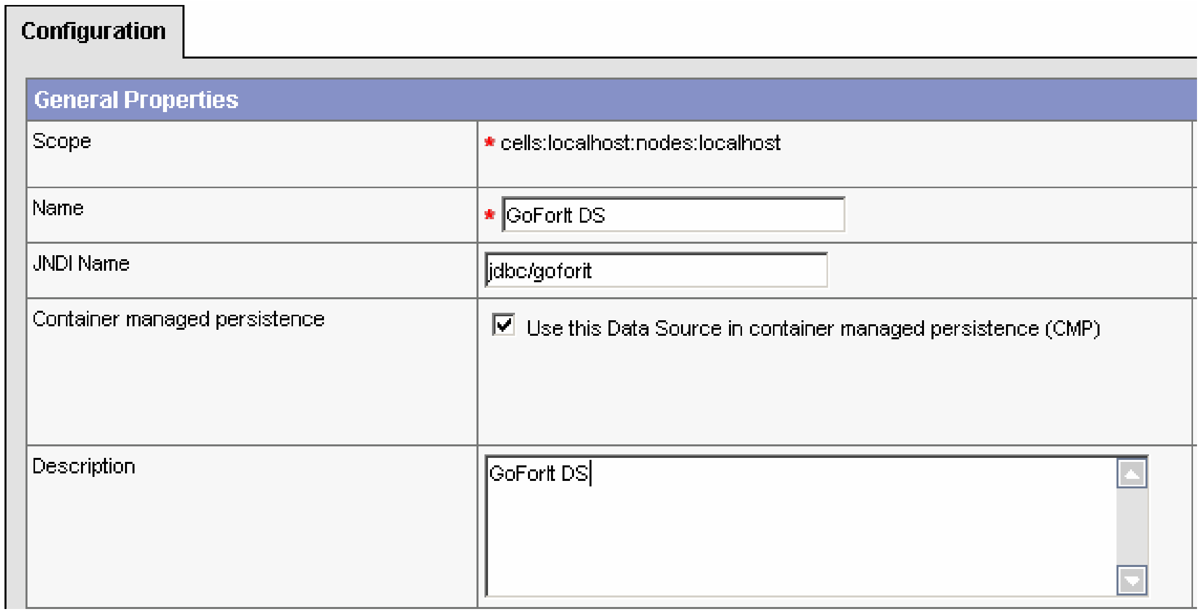
__i. The Configuration for the **DB2 JDBC Provider (XA**) will be displayed again. Below it, you can set additional properties. Click **Data Sources** (note: you may have to scroll down to see it)

__j. The empty list of Data Sources for this newly created provider will be displayed. Click **New** to create a Data Source.

__k. The configuration for the new Data Source is displayed. Enter the following:

Name:	**GoForIt DS**
JNDI Name:	**jdbc/goforit**
Description:	**GoForIt DS**

Check the checkbox labeled **Use this Data Source in container managed persistence (CMP)**



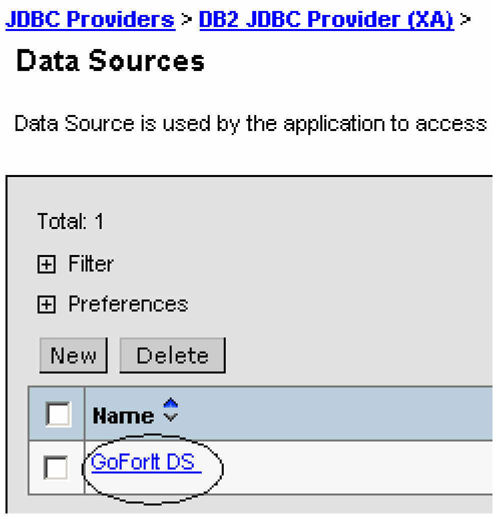
Configuration

General Properties

Scope	* cells:localhost:nodes:localhost
Name	* GoForIt DS
JNDI Name	jdbc/goforit
Container managed persistence	☑ Use this Data Source in container managed persistence (CMP)
Description	GoForIt DS

__l. Leave all other values at their defaults and click **OK**

__m. The Data Source is created , but there is one more configuration property that needs to be set. Click **GoForIt DS**



JDBC Providers > DB2 JDBC Provider (XA) >

Data Sources

Data Source is used by the application to access

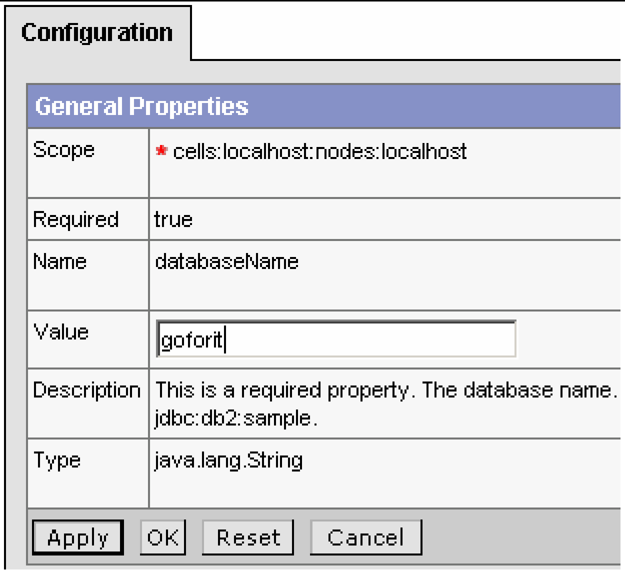
Total: 1

	Name
☐	GoForIt DS

__n. The Data Source properties will be shown again. Below the configuration, additional properties can be set. Click **Custom Properties**

__o. Click **databaseName**

__p. Change the Value to **goforit** and click **OK**



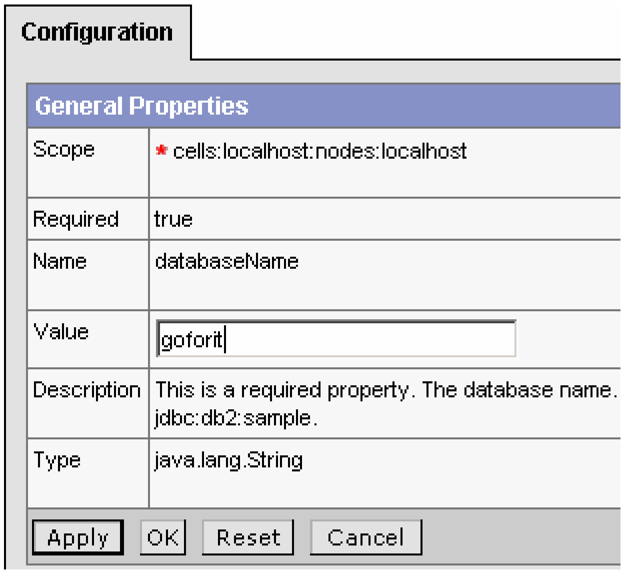
Configuration

General Properties

Scope	* cells:localhost:nodes:localhost
Required	true
Name	databaseName
Value	goforit
Description	This is a required property. The database name. jdbc:db2:sample.
Type	java.lang.String

__3. Your Data Source is now defined. With WebSphere Application Server v5.0, any configuration changes made via the Admin console must be explicitly saved.

__a. Click **Save** from the header at the upper left of the page

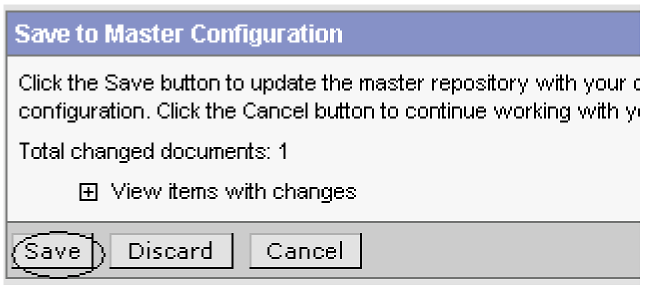


Configuration

General Properties

Scope	* cells:localhost:nodes:localhost
Required	true
Name	databaseName
Value	goforit
Description	This is a required property. The database name. jdbc:db2:sample.
Type	java.lang.String

__b. The Save page will be displayed. Click **Save**



Save to Master Configuration

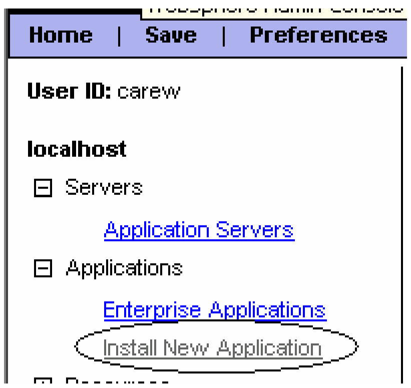
Click the Save button to update the master repository with your c
configuration. Click the Cancel button to continue working with y

Total changed documents: 1

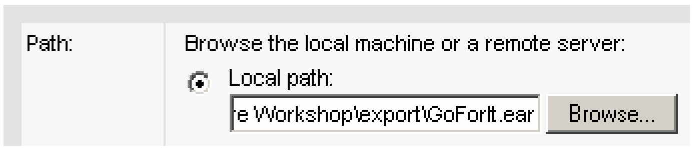
⊞ View items with changes

__4. With the Data Source taken care of, it's time to install the Enterprise Application

__a. Expand the **Applications** tree and click on **Install New Application**



__b. Click **Browse** for the local path and navigate to **C:\WebSphere Workshop\export\GoForIt.ear**

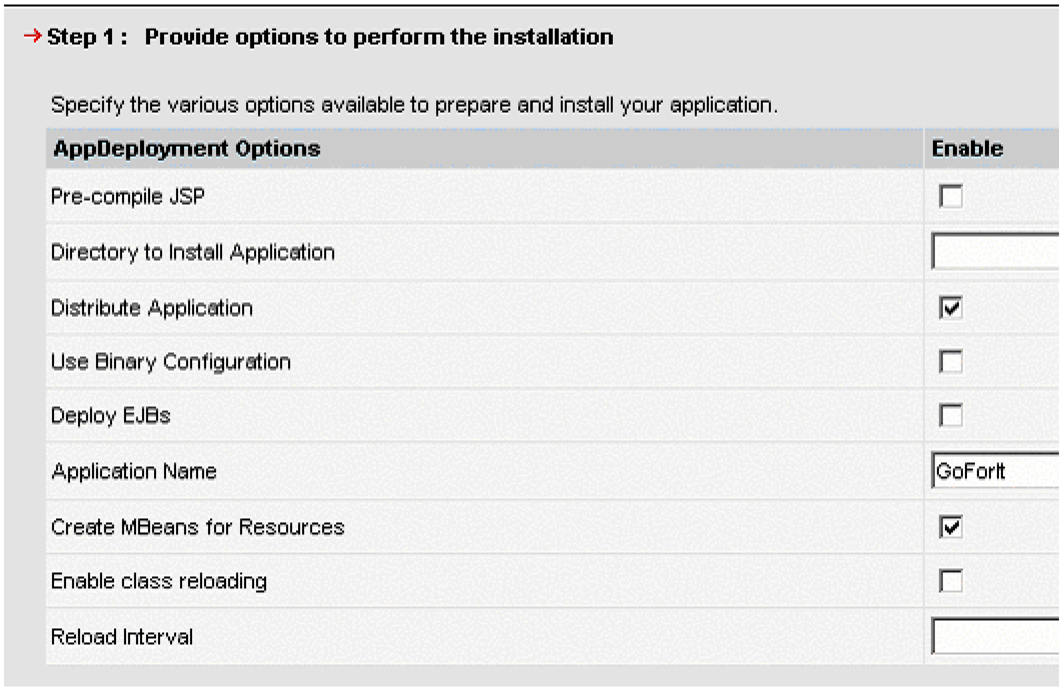


Path:	Browse the local machine or a remote server:
	● Local path: e Workshop\export\GoForIt.ear Browse...

__c. Click **Next**

__d. You can accept the default bindings (specified in the .ear file) by clicking **Next** again

__e. Next you will presented with a series of installation steps. If you're curious, you can click on each step to see what options can be changed

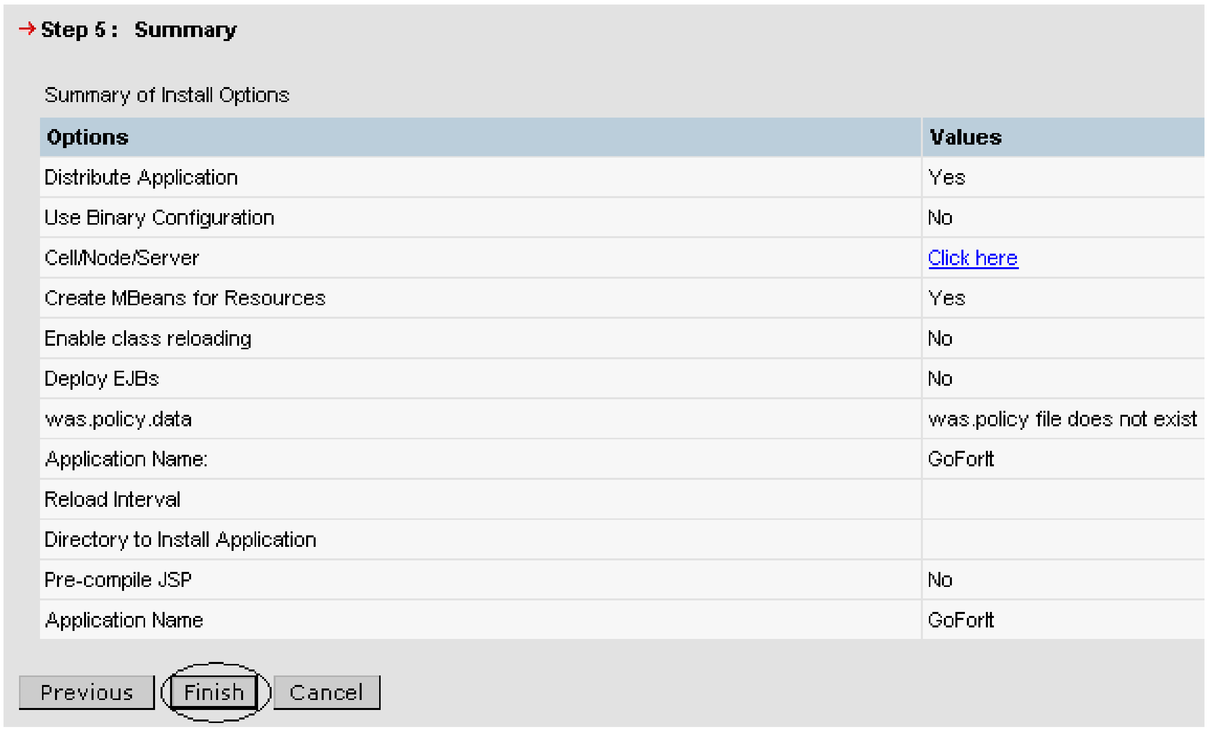


→Step 1 : Provide options to perform the installation

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	☐
Directory to Install Application	
Distribute Application	✓
Use Binary Configuration	☐
Deploy EJBs	☐
Application Name	GoForIt
Create MBeans for Resources	✓
Enable class reloading	☐
Reload Interval	

__f. When you're done looking around, click on Step 5 and then click on Finish to complete the installation



→ Step 5 : Summary

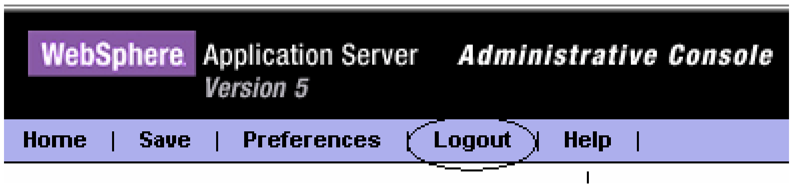
Summary of Install Options

Options	Values
Distribute Application	Yes
Use Binary Configuration	No
Cell/Node/Server	Click here
Create MBeans for Resources	Yes
Enable class reloading	No
Deploy EJBs	No
was.policy.data	was.policy file does not exist
Application Name:	GoForIt
Reload Interval	
Directory to Install Application	
Pre-compile JSP	No
Application Name	GoForIt

__g. Save the changes by clicking on **Save** in the header and **Save** in the Save Page as you did after creating the Data Source

__5. Exit the Admin Console

__a. Exit the Admin Console by clicking on **Logout**


WebSphere Application Server Version 5
Administrative Console
Home | Save | Preferences | Logout | Help |

__b. Shutdown down your browser

__6. Regenerate the plugin configuration for the IBM HTTP Server and restart the server the realize the configuration changes

__a. Go back to the command window that you used to start the server and type

genplugincfg

__b. When the command completes, stop the server by entering the following command

stopserver server1

__c. When the stopserver commands completes, start the server again by entering the following command

startserver server1

Part Three: Testing the Application

__1. Test the application

__a. Start your browser and go to the URL **http://localhost/GoForItWeb/index.jsp** . Verify that the login page comes up.

__b. Login with userid **joeg** and a password of **password**. Verify that the Main menu page comes up.

Part Four: Clean up

__1. You'll be deploying a different version of the GoForIt application tomorrow so uninstall this one now using the the wsadmin command line interface

__a. From the command prompt that you have been using to start and stop the server enter the following command

wsadmin –c "$AdminApp uninstall GoForIt"

This will remove the GoForIt Enterprise Application from the server configuration

__b. Stop the server

C:\Program Files\WebSphere\AppServer\bin>stopserver server1

What you did in this exercise

In this lab you added a JDBC Provider and a Data Source to the WebSphere 5.0 server configuration using the Admin Console. You then installed the GoForIt application and tested it.

Creating and using Container Managed Persistence Entity Beans

What this exercise is about

In this exercise you will learn how to build and use Container Managed Persistence EJBs. The GoFor-It User EJB will be used for managing users and the Errand EJB will be used to manage the errands that are requested by users.

What You Should Be Able To Do

After completing this lab, you will be familiar with creating and testing CMP Entity Beans. You will have learned to write ejbCreate methods as well as ejbSelect and finder methods. You will also be able to perform simple mappings between java objects and database tables using Application Developer.

Introduction

This exercise uses one of three different methods that are available to developers using Application Developer to map CMP Entity Beans to the database. The method that will be used is “Bottom-up” where Entity Beans are generated based on an existing database schema.

The EJBs will be generated with Local interfaces only, since it is assumed that Session Beans in the same container will be used to access the Entity beans, and thus there is no reason to make the Entity beans accessible outside the JVM in which the container is running.

The lab will walk you through the process of developing and then testing EJBs using Application Developer which has a lot of built-in support for the entire EJB Development cycle. The lab will also demonstrate how custom queries can be written in EJB-QL and then exposed to the internal components of the bean or to clients of the bean as finder methods.

Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__1. Start WebSphere Studio Application Developer

__a. From the Windows Start Menu select **Start>Programs-IBM WebSphere Studio>Application Developer**

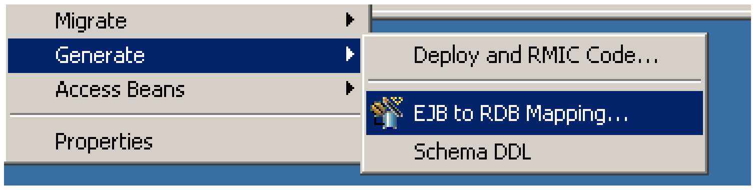
__b. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab4**

__c. Click **OK**

Part One: Creating the User and Errand EJBs

__1. An empty EJB project has already been created for you. Start by generating the EJBs based on the existing database schema that we've used in the previous labs.

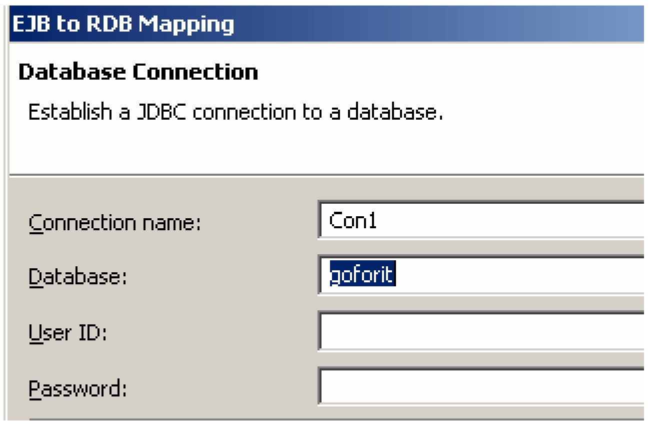
__a. From the J2EE Navigator view, select the the **GoForItEJB** project, right click and select **Generate>EJB to RDB Mapping**



__b. Make sure **Create a new backend folder** is selected and click **Next**

__c. Since the EJB Project is empty only **Bottom Up** can be selected as the the type of mapping . Click **Next**

__d. In the connection dialog, enter **goforit** as the database name and click **Next**


EJB to RDB Mapping

Database Connection

Establish a JDBC connection to a database.

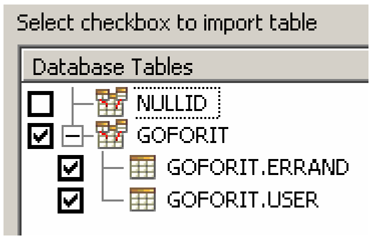
Connection name: Con1

Database: goforit

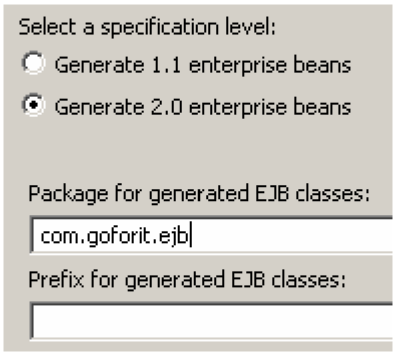
User ID:

Password:

__e. Select the checkbox next to the GOFORIT database to select the 2 tables User and Errand. Click **Next**



__f. Make sure **Generate 2.0 enterprise beans** is selected and enter **com.goforit.ejb** as the package name. Click **Finish**


Select a specification level:
○ Generate 1.1 enterprise beans
● Generate 2.0 enterprise beans

Package for generated EJB classes: com.goforit.ejb

Prefix for generated EJB classes:

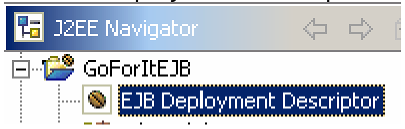
The EJBs are generated and the Mapping Editor opens displaying the mappings between the generated EJB fields and the columns in the database tables

Part Two: Creating the EJBQL queries and the custom create method

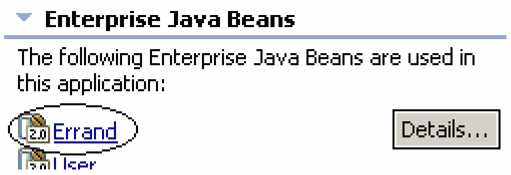
The Primary Key for the User EJB is the userid supplied by the user that is registering. If the user supplies a userid that is already in use, they will be advised that it is a duplicate and be required to supply a unique value. For the Errand EJB the primary key is an integer. It is not reasonable to expect the requester of an errand to supply a unique errand id so the approach used in the creation of a new errand is to find out the largest existing errand id and increment it by one.

__1. Since EJB creation is done within the bean class, we can write an EJBQL ejbSelect method to find the largest existing errand id and increment it by one to get a new errand id when creating an errand. EJBQL for ejbSelects and finders is defined in the deployment descriptor

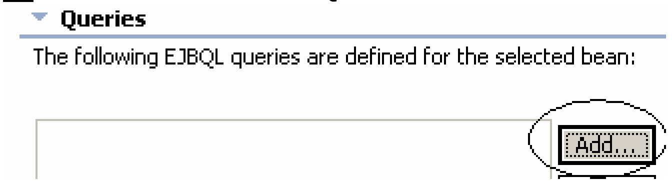
__a. In the J2EE Navigator view, expand the **GoForItEJB** project and select EJB Deployment Descriptor. Double click on it to bring it up in its editor.



__b. In the Enterprise Java Beans section, click on **Errand**

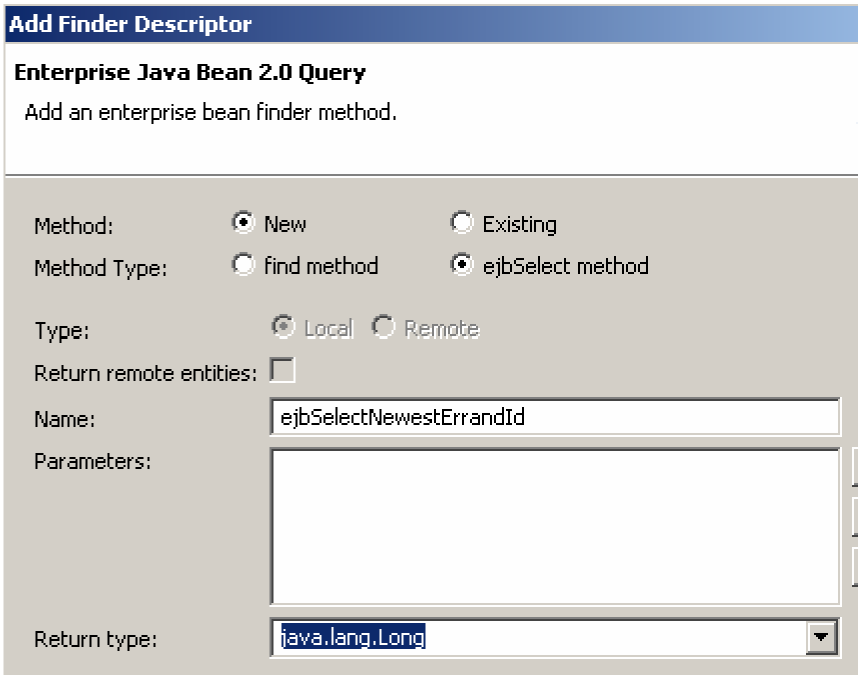

Enterprise Java Beans
The following Enterprise Java Beans are used in this application:
Errand
User
Details...

__c. Scroll down to the **Queries** section and click on Add


Queries
The following EJBQL queries are defined for the selected bean:
Add...

__d. Use the following values in the Add Finder Descriptor dialog

Method:	**New**
Method Type:	**ejbSelect**
Name:	**ejbSelectNewestErrandId**
Return Type	**`java.lang.Long`**


Add Finder Descriptor

Enterprise Java Bean 2.0 Query

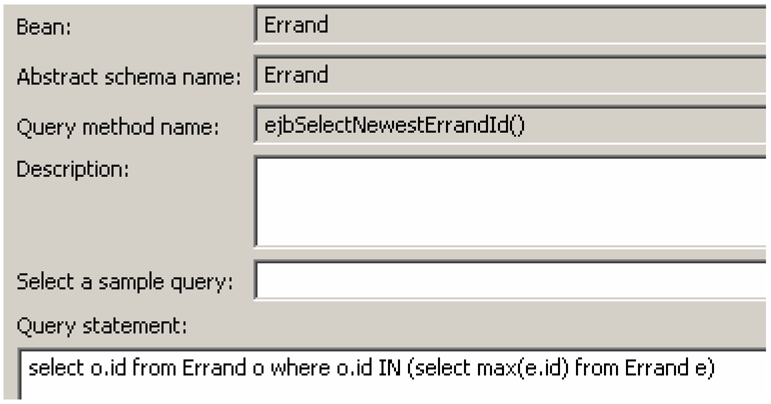
Add an enterprise bean finder method.

Method: [✓ New] Existing
Method Type: find method [✓ ejbSelect method]
Type: [✓ Local] Remote
Return remote entities: ☐
Name: ejbSelectNewestErrandId
Parameters:
Return type: java.lang.Long

__e. Click **Next**

__f. Enter the following as the Query Statement and click **Finish**

select o.id from Errand o where o.id IN (select max(e.id) from Errand e)


Bean: Errand
Abstract schema name: Errand
Query method name: ejbSelectNewestErrandId()
Description:
Select a sample query:
Query statement:
select o.id from Errand o where o.id IN (select max(e.id) from Errand e)

__g. Enter **Ctrl+S** to save you changes. Keep file open for the next step.

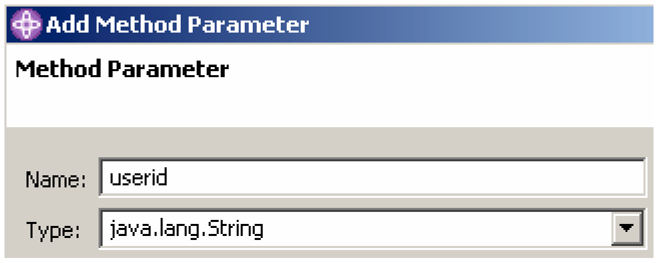
__2. Next we'll do a finder that will return all the errands for a given userid.

__a. Click on **Add** next to the Queries section again

__b. Add the following values

Method:	**New**
Method Type:	**find method**
Name:	**findByUser**
Return Type	**`java.util.Collection`**

__c. Click on the **Add** button next to **Parameters** and enter **userid** as the **Name** and **java.lang.String** as the **Type**

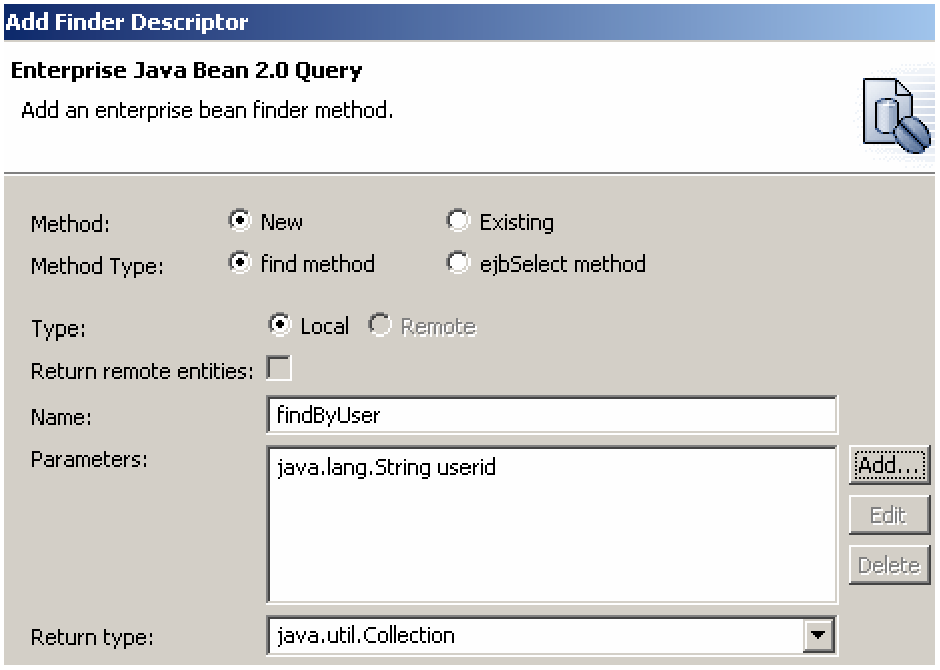

Add Method Parameter

Method Parameter

Name: userid

Type: java.lang.String

__d. Click **OK**


Add Finder Descriptor

Enterprise Java Bean 2.0 Query

Add an enterprise bean finder method.

Method: [✓ New] Existing

Method Type: [✓ find method] ejbSelect method

Type: [✓ Local] Remote

Return remote entities: ☐

Name: findByUser

Parameters: java.lang.String userid

Return type: java.util.Collection

__e. Click **Next**

__f. Enter the following as the Query Statement and click **Finish**

select object(o) from Errand o where o.requester = ?1

__g. Enter **Ctrl+S** to save your changes

__3. The code that will interface with our EJBs will use the ErrandDataBean class (a simple Java Bean) to represent an errand . It will be convenient if our create method for the Errand EJB accepts an instance of ErrandDataBean. We also need to implement our scheme for generating the errand id automatically. We will do both in a create method for the Errand EJB.

__a. Create methods are defined on the actual bean and then promoted - added to - to the Local Home Interface. From the J2EE Navigator view in the J2EE Perspective expand **GoForItEJB**, then expand **ejbModule** and drill down to the **com.goforit.ejb** package and double click on **ErrandBean.java**

__b. **Add** the following method (**Note**: you can copy this from the file C:\WebSphere Workshop\snippets\lab4\ejbCreate.txt).

```
/**
 * ejbCreate method for a CMP entity bean.
 */
public ErrandKey ejbCreate(ErrandDataBean newErrand)
       throws javax.ejb.CreateException {
       try {
          setId(getNextErrandId());
          setCategory(newErrand.getCategory());
          setSubcategory(newErrand.getSubcategory());
          setStatus(newErrand.getStatus());
          setDescription(newErrand.getDescription());
          setDuedate(new java.sql.Date(newErrand.getDueDate().getTime()));
          setRequester(newErrand.getRequester());
       } catch (Exception e) {
          throw new CreateException(e.getMessage());
       }
       return null;
}
```

__c. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__d. Format the code you just added by entering **Ctrl+Shift+F**

__e. You'll get an error because the method getNextErrandId() hasn't been defined yet. This private method will use the ejbSelect you defined previously. Add the following method (**Note**: you can copy this from the file C:\WebSphere Workshop\snippets\lab4\ejbCreate.txt).

```
/**
 * Returns the next errand id that can be used to add
 * a new errand
 */
private Long getNextErrandId() {
       Long newestId = null;
       try {
               newestId = ejbSelectNewestErrandId();
       }
       catch (FinderException e) {
              return new Long(1);
       }
       return new Long(newestId.longValue() + 1);
}
```

__f. The next thing to do is to add a matching **ejbPostCreate** method. The EJB Specification requires that every **ejbCreate** must have a matching (with the same parameter list) **ejbPostCreate** method. Add the following method to **ErrandBean.java**

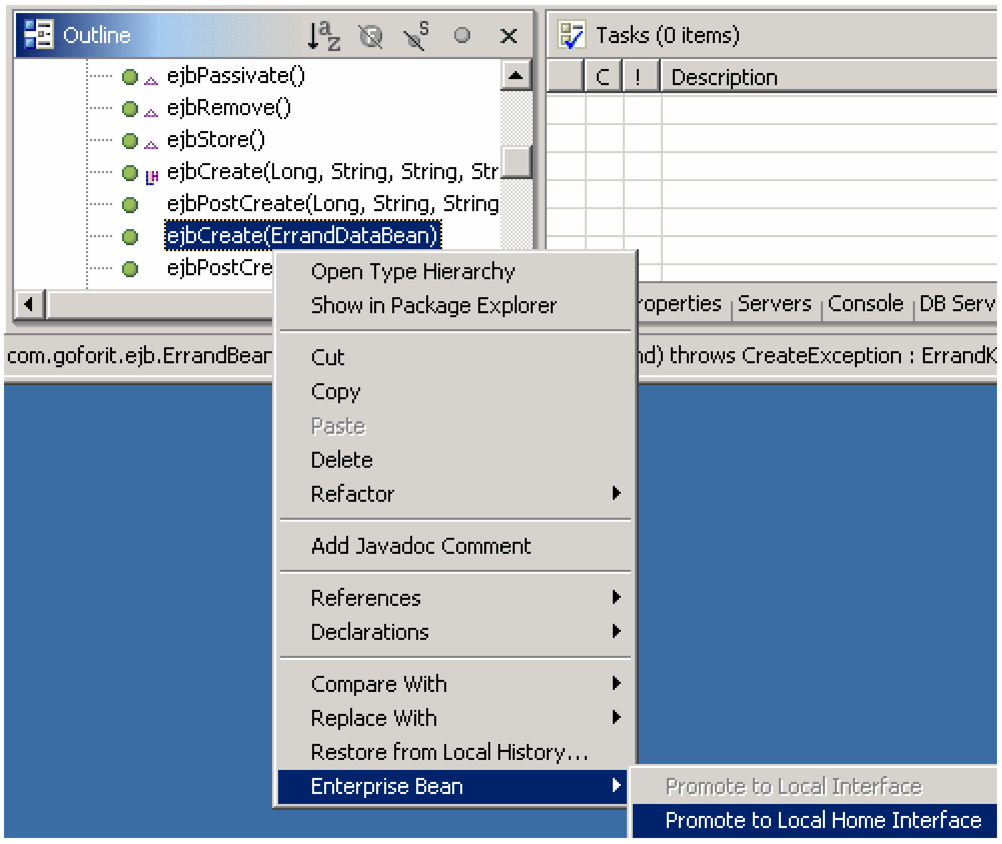
```
public void ejbPostCreate(ErrandDataBean errand)
        throws javax.ejb.CreateException {
}
```

__g. Add a convenience method to get the Primary Key field from the Local Interface. Add the following code

```
public long getErrandId()  {
   return getId().longValue();
}
```

__h. Enter **Ctrl+S** to save your changes

__i. Promote the ejbCreate() method that you just added to the Local Home Interface so it will be accessible by clients of the EJB. In the Outline view select the method, right click and select **Enterprise Bean>Promote to Local Home Interface**



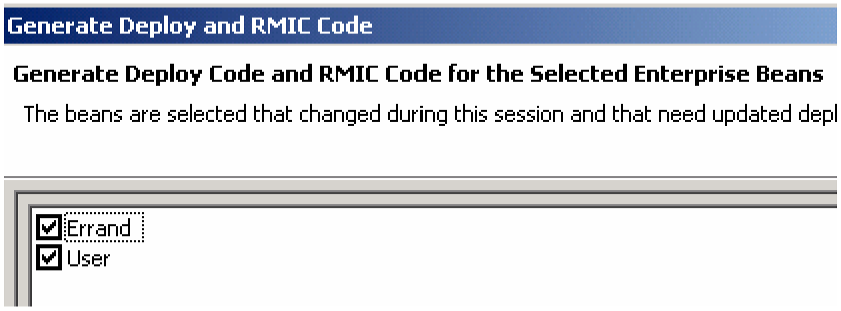
__j. Promote the getErrandId() method to the Local Interface. In the Outline view select the method, right click and select **Enterprise Bean>Promote to Local Interface**

Part Three: Generating the Deploy code

__1. Your EJBs are now ready to be tested. However, before you can actually run them you must generate the deployed code (the code that the EJB container generates when the EJBs are deployed to a server).

__a. Go to the J2EE Navigator View in the J2EE Perspective, and select the **GoForItEJB** project. Right-click and select **Generate->Deploy and RMIC code**.

__b. Make sure both Errand and User are selected and click **Finish**


Generate Deploy and RMIC Code

Generate Deploy Code and RMIC Code for the Selected Enterprise Beans

The beans are selected that changed during this session and that need updated depl

☑ Errand
☑ User

__c. This process takes a few seconds, a bit longer the first time it runs. Look in the J2EE Navigator view and you will discover that there are now an additional number of classes and interfaces in the **com.goforit.ejb** package. Feel free to look around at the generated code.

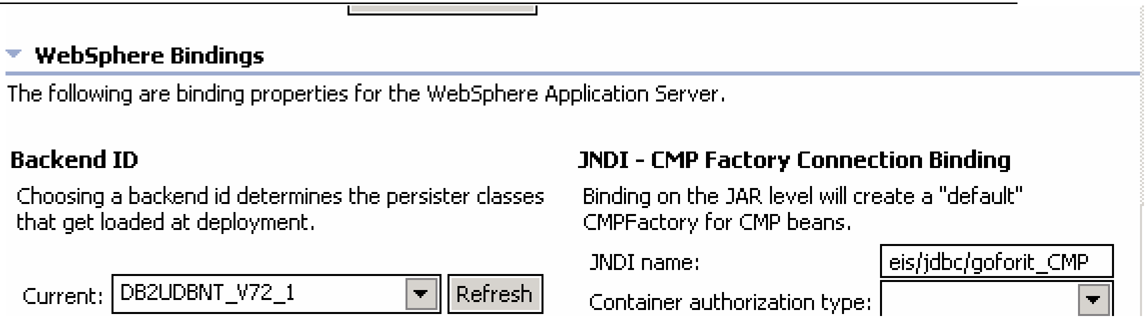
Part Four: Testing the EJBs

The WebSphere Test Environment within Application Developer allows you to test your EJBs on a server running inside Application Developer

__1. Before the EJBs can be tested, they must be mapped to the Data source that you have been using with the previous labs. This loose coupling between EJBs and databases allows you to switch databases without having to modify your code. The mapping is done in the EJB Deployment Descriptor. WebSphere 5.0 uses a special JCA resource adapter that runs on top of JDBC, so we have to map the EJBs to the resource adapter instead of to a JDBC Data source directly. Each Data source will have a corresponding resource adapter in WebSphere.

__a. In the J2EE Navigator view, expand the **GoForItEJB** project and select EJB Deployment Descriptor. Double click on it to bring it up in its editor.

__b. Scroll down to the very bottom to the WebSphere Bindings section and select **DB2UDBNT_V72_1** as the Current Binding and enter **eis/jdbc/goforit_CMP** as the JNDI Name



WebSphere Bindings

The following are binding properties for the WebSphere Application Server.

Backend ID

Choosing a backend id determines the persister classes that get loaded at deployment.

Current: DB2UDBNT_V72_1 Refresh

JNDI - CMP Factory Connection Binding

Binding on the JAR level will create a "default" CMPFactory for CMP beans.

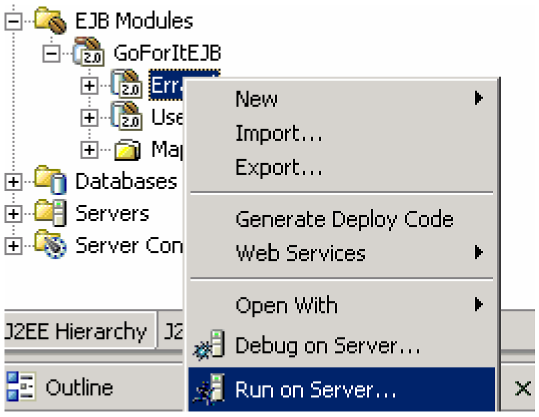
JNDI name: eis/jdbc/goforit_CMP

Container authorization type:

__c. Enter **Ctrl+S** to save your changes

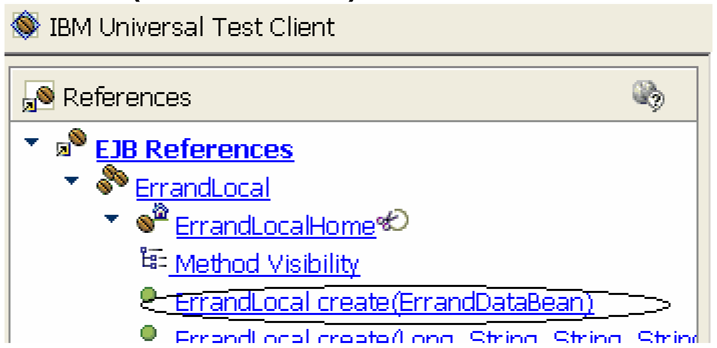
__2. Next you'll launch the Test Client Web application to test the Errand EJB

__a. In the J2EE Hierarchy view, expand **EJB Modules, GoForItEJB** and then select **Errand**. Right click and select **Run on Server**

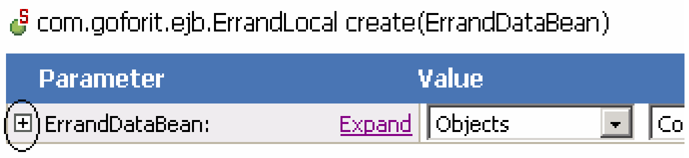


__b. This will launch the Test Client in your browser after starting the server. Accept the default **Use an existing server** and click on **Finish**

__c. When the browser comes up go to the References frame and expand **ErrandLocal** , then **ErrandLocalHome** and click on **ErrandLocal create(ErrandDataBean)**


IBM Universal Test Client
References
EJB References
ErrandLocal
ErrandLocalHome
Method Visibility
ErrandLocal create(ErrandDataBean)
ErrandLocal create(long, String, String, Strin

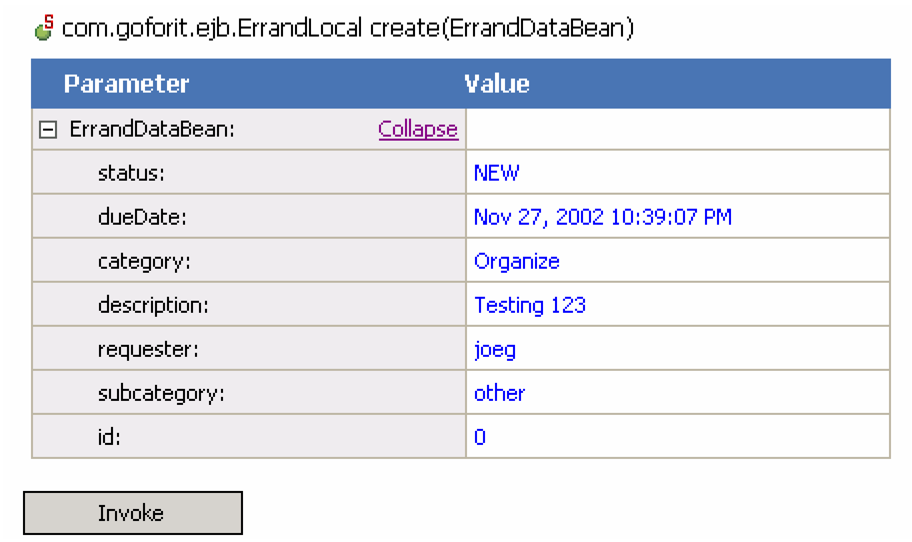
__d. In the Parameters frame on the right, expand ErrandDataBean


com.goforit.ejb.ErrandLocal create(ErrandDataBean)

Parameter	Value
ErrandDataBean: Expand	Objects Co

__e. Enter the following values

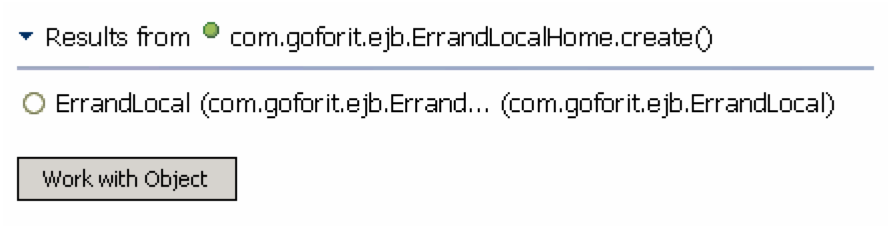
status	**NEW**
category	**Organize**
description	**Testing 123**
requester	**joeg**
subcategory	**other**


com.goforit.ejb.ErrandLocal create(ErrandDataBean)

Parameter	Value
⊟ ErrandDataBean: Collapse	
status:	NEW
dueDate:	Nov 27, 2002 10:39:07 PM
category:	Organize
description:	Testing 123
requester:	joeg
subcategory:	other
id:	0

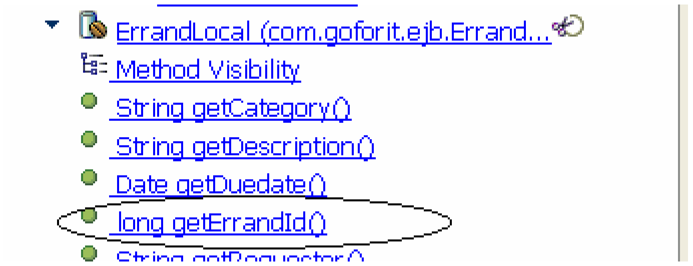
__f. Click on **Invoke**

__g. The result is returned right below. Click on Work with Object

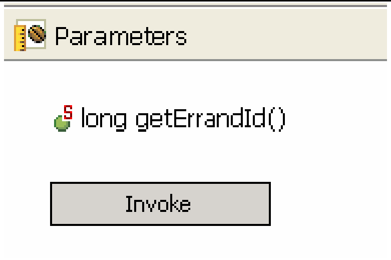

Results from com.goforit.ejb.ErrandLocalHome.create()

○ ErrandLocal (com.goforit.ejb.Errand... (com.goforit.ejb.ErrandLocal)

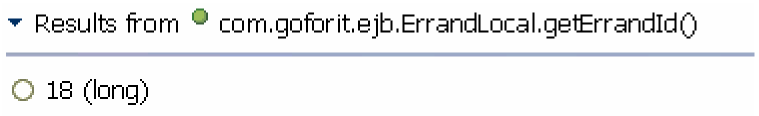
__h. Select the getErrandId()method from the ErrandLocal instance that is now on the left frame



__i. On the right click on **Invoke**



__j. Note the id that's returned just below



Results from com.goforit.ejb.ErrandLocal.getErrandId()

18 (long)

__3. We can verify that the scheme for generating the errand id has worked correctly by going directly to the database

__a. From the Windows Start prompt select

Start>Programs>IBM DB2->Command Line Processor

__b. At the prompt, type **connect to goforit**

__c. At the next prompt type **select id from goforit.errand**

```
db2 => connect to goforit

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = CAREW
 Local database alias   = GOFORIT

db2 => select id from goforit.errand

ID
--------------------
                   1
                   2
                   3
                   4
                   5
                   6
                   7
                   8
                   9
                  10
                  11
                  12
                  13
                  14
                  15
                  16
                  17
                  18

  18 record(s) selected.
```

__d. Verify that the largest id is the same as the one you got back from the call to getErrandId() in the Test Client

__4. Shutdown the server and Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop**

__c. Shutdown Application Developer (**File->Exit**)

What you did in this exercise

In this exercise you created two Container Managed Enterprise JavaBeans by using the EJB Tools within Application Developer to import an existing schema. You also defined custom finder and ejbSelect methods using EJBQL and a custom create methods and tested the EJBs using the Test Client in Application Developer.

IBM WebSphere Application Server v5.0 Workshop – LAB EXERCISE

Stateless Session Beans and Message Driven Beans

What this exercise is about

In this lab you will learn how to create and test a Stateless Session Enterprise JavaBean and a Message Driven Bean. You will use Application Developer to perform these tasks.

What You Should Be Able To Do

After completing this lab, you will be familiar with creating and testing Session Beans and Message Driven Beans.

Introduction

In this lab you will create the CustomerController EJB from scratch. CustomerController will act as a façade to the Entity beans you created earlier. You will create the bean using Application Developer. You will then proceed to add the business methods to the bean. After the business methods are defined, you will add the methods to the Local Interface. You will generate the necessary support classes and using Application Developer's Test Client, you will test and debug your EJB. After creating and testing the EJB, you will use it for the Go-ForIt website.

The EJB will be generated with Local interfaces only, since it is assumed that the clients of the Session Bean will be the web components running in the same JVM as the EJB .

After creating the Session Bean, you'll create a Message Driven bean that responds to messages on a designated queue and then invokes the Session Bean you created to process an errand request. The JMS provider that will be used is the Embedded JMS Provider that comes with WebSphere 5.0.

After creating the Message Driven you'll test it by running a J2EE Client Application that will place a message on the queue that the MDB is associated with, and then you'll verify that the errand request is successfully handled.

Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__1. Start WebSphere Studio Application Developer

__a. From the Windows Start Menu select **Start>Programs-IBM WebSphere Studio>Application Developer**

__b. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab5**

__c. Click **OK**

__d. ***Note: You'll have some errors because the code to interface with the CustomerController Session Bean has already been written***

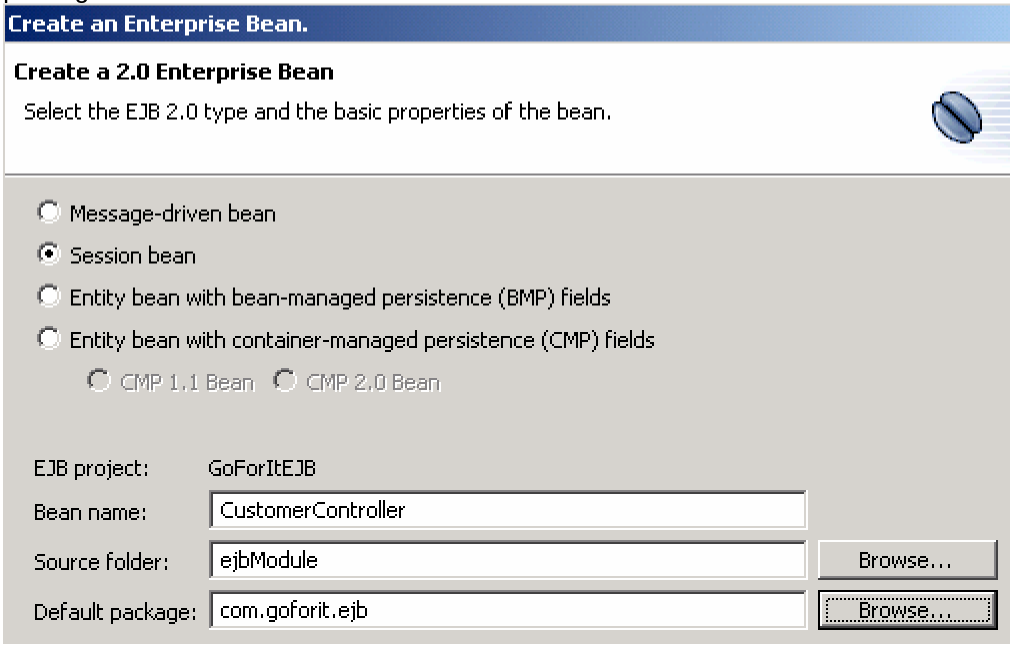
Part One: Creating the CustomerController EJB

__1. Start by bringing up the EJB wizard to create the CustomerController EJB.

__a. From the J2EE Hierarchy view, expand **EJB Modules**, select **GoForItEJB** , right click and select **New>Enterprise Bean**

__b. Click **Next** in the Project Selection Dialog

__c. Select **Session Bean**, type **CustomerController** as the bean name and click the **Browse** button next to **Default Package** to select **com.goforit.ejb** as the default package


Create an Enterprise Bean.

Create a 2.0 Enterprise Bean

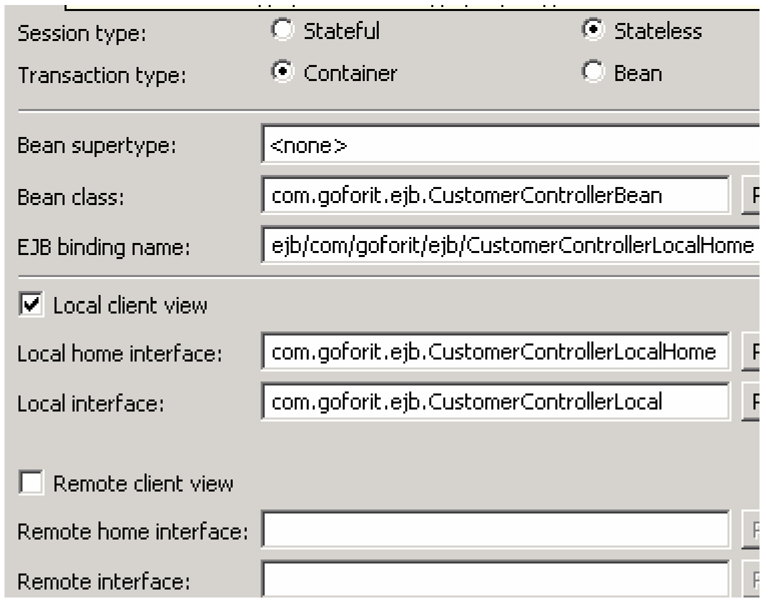
Select the EJB 2.0 type and the basic properties of the bean.

( ) Message-driven bean
(●) Session bean
( ) Entity bean with bean-managed persistence (BMP) fields
( ) Entity bean with container-managed persistence (CMP) fields
( ) CMP 1.1 Bean ( ) CMP 2.0 Bean

EJB project: GoForItEJB
Bean name: CustomerController
Source folder: ejbModule
Default package: com.goforit.ejb

__d. Click **Next**

__e. Uncheck **Remote Client View** and check **Local Client View** (we'll only be using Local Interfaces with this EJB)


Session type: ○ Stateful ● Stateless
Transaction type: ● Container ○ Bean
Bean supertype: <none>
Bean class: com.goforit.ejb.CustomerControllerBean
EJB binding name: ejb/com/goforit/ejb/CustomerControllerLocalHome
☑ Local client view
Local home interface: com.goforit.ejb.CustomerControllerLocalHome
Local interface: com.goforit.ejb.CustomerControllerLocal
☐ Remote client view
Remote home interface:
Remote interface:

__f. Click **Finish**

__2. The next step is to add the business methods to the CustomerController bean you just created

__a. In the J2EE Navigator view expand **GoForItEJB**, then **ejbModule** and then **com.goforit.ejb** and double click on **CustomerControllerBean.java** to bring it up in the Java Editor

__b. Replace the contents of the class with the code from C:\WebSphere Workshop\snippets\lab5\customerController.txt

__c. Read through the code to make sure you understand what each method is doing

__d. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__e. Format the code you just added by entering **Ctrl+Shift+F**

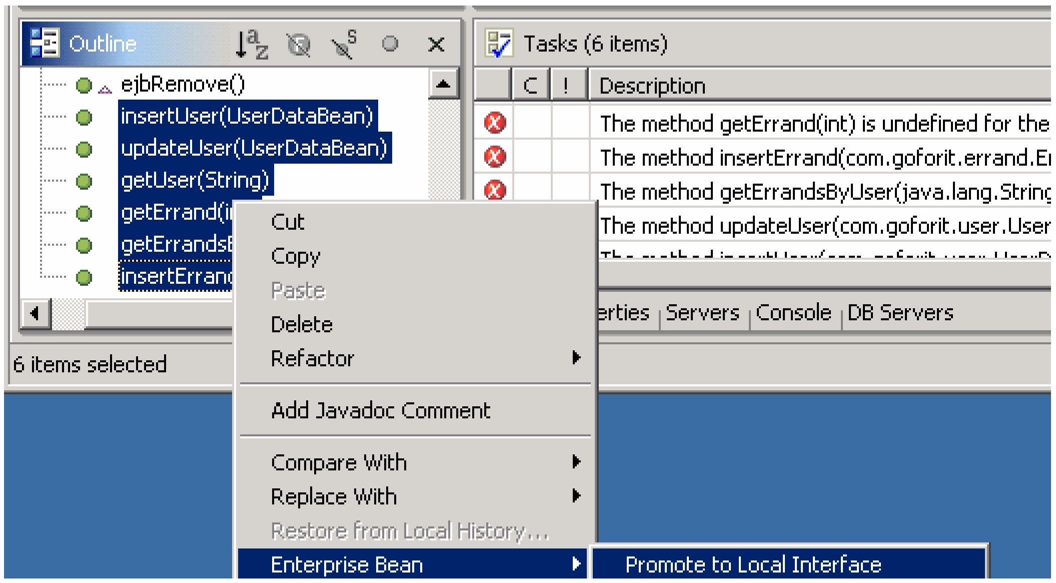
__f. Save the file by entering **Ctrl+S**

__3. The added business methods need to be added to the Local interface so they are accessible from the web container code that uses them

__a. Select the following methods from the **CustomerControllerBean** class in the **Outline** pane. (Hint: you can select multiple methods by pressing the **Ctrl** key while selecting methods).

```
insertErrand(ErrandDataBean)
getErrand(int)
getErrandsByUser(String)
getUser(String)
insertUser(UserDataBean)
updateUser(UserDataBean)
```

__b. Right click and select **Enterprise Bean>Promote to Local Interface**



The method getErrand(int) is undefined for the
The method insertErrand(com.goforit.errand.E
The method getErrandsByUser(java.lang.String
The method updateUser(com.goforit.user.User

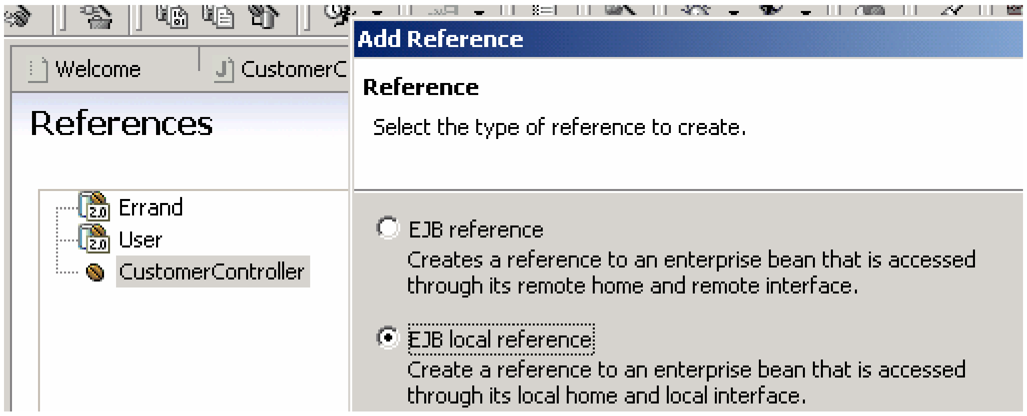
__4. To complete the Session Bean we'll add EJB references to the Errand and User EJBs so their JNDI names won't have to be hard coded inside our session bean

__a. In the J2EE Navigator view, expand the **GoForItEJB** project and select **EJB Deployment Descriptor**. Double click on it to bring it up in its editor.

__b. Click on the **References** tab at the bottom of the editors window

__c. Under **References,** select **CustomerController** and click on **Add**

__d. Select **EJB local reference** and click on **Next**



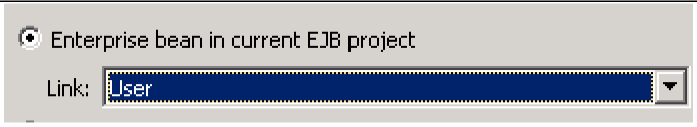
Select the type of reference to create.

( ) EJB reference
Creates a reference to an enterprise bean that is accessed through its remote home and remote interface.

(•) EJB local reference
Create a reference to an enterprise bean that is accessed through its local home and local interface.

__e. Click on the **Browse** button next to **Link** and select the **User** EJB in the current project

__f. Click **OK**



__g. Click **Finish**

__h. Click **Add** again and add a reference for the Errand EJB by repeating steps d through g for Errand instead of User

__i. Enter **Ctrl+S** to save your changes and then close the editor's window

Part Two: Generating the Deploy code

__1. Your EJB is now ready to be tested. However, before you can actually test it you must generate the deployed code (the code that the EJB container generates when the EJB is deployed to a server).

__a. Go to the J2EE Navigator View in the J2EE Perspective, and select the **GoForItEJB** project. Right-click and select **Generate->Deploy and RMIC code**.

__b. Select CustomerController (you can leave both Errand and User unchecked) and click **Finish**

Part Three: Test the Session Bean

Since the code to interface with the Session Bean has already been written, this can be used to test it. Of course, we could also have used the Test Client that was used for the Entity beans in the previous lab

__1. Start the WebSphere 5.0 Test Environment in Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Start**. Wait for the message in the console that says
Server server1 open for e-business before continuing

__2. Test the Session bean by running through a couple of scenarios

__a. Start your browser and go to the URL **http://localhost:9080/GoForItWeb/index.jsp**.

__b. Login with userid **joeg** and a password of **password**. Verify that the Main menu page comes up.

__c. Click on the **View all errands** link and verify that a list of errands is displayed along with a link to view the details of each errand

__d. Click on **View Details** for one of the errands and verify that details are displayed.

__3. Shutdown the server

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop**

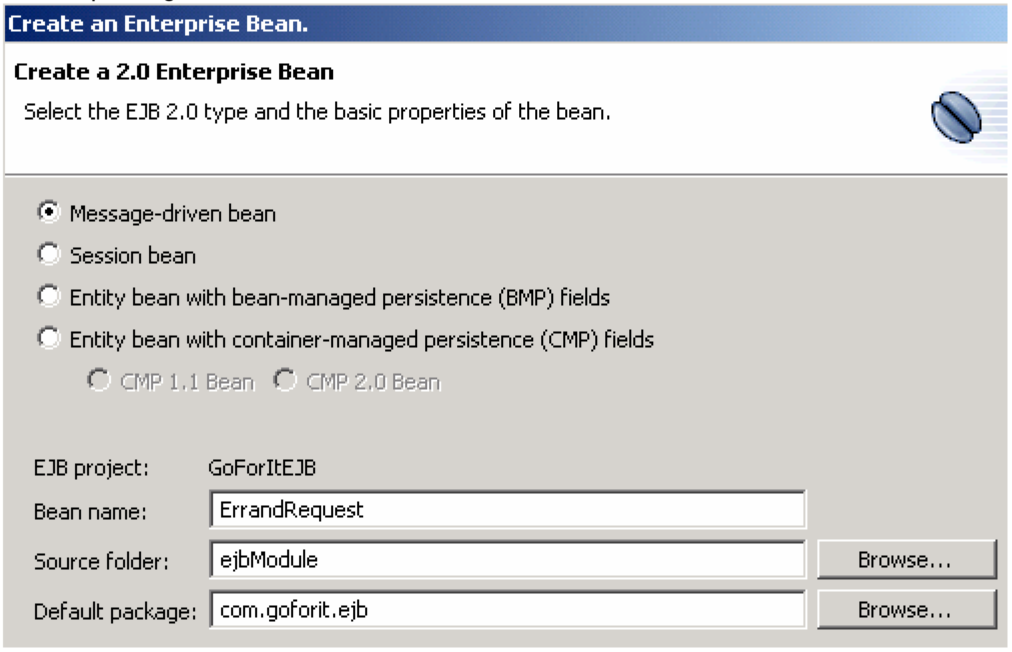
Part Four: Creating the Message Driven Bean

__1. Start by bringing up the EJB wizard to create the Message Driven Bean.

__a. From the J2EE Hierarchy view, expand **EJB Modules**, select **GoForItEJB** , right click and select **New>Enterprise Bean**

__b. Click **Next** in the Project Selection Dialog

__c. Select **Message-driven Bean**, type **ErrandRequest** as the bean name and click the **Browse** button next to **Default Package** to select **com.goforit.ejb** as the default package



Create an Enterprise Bean.

Create a 2.0 Enterprise Bean

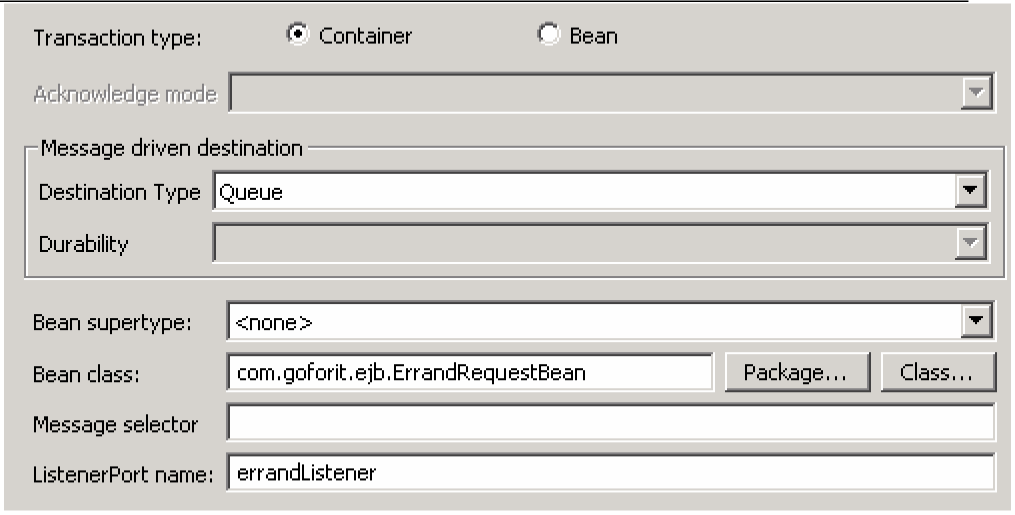
Select the EJB 2.0 type and the basic properties of the bean.

[✓ Message-driven bean]
Session bean
Entity bean with bean-managed persistence (BMP) fields
Entity bean with container-managed persistence (CMP) fields
CMP 1.1 Bean CMP 2.0 Bean

EJB project: GoForItEJB
Bean name: ErrandRequest
Source folder: ejbModule
Default package: com.goforit.ejb

__d. Click **Next**

__e. Select **Queue** as the **Destination Type** and type in **errandListener** as the **ListenerPort Name**


Transaction type: [✓ Container] Bean
Acknowledge mode
Message driven destination
Destination Type: Queue
Durability
Bean supertype: <none>
Bean class: com.goforit.ejb.ErrandRequestBean

Message selector
ListenerPort name: errandListener

__f. Click **Finish**

__2. The next step is to add the business methods to the CustomerController bean you just created

__a. In the J2EE Navigator view expand **GoForItEJB**, then **ejbModule** and then **com.goforit.ejb** and double click on **ErrandRequestBean.java** to bring it up in the Java Editor

__b. Replace the contents of the class with the code from C:\WebSphere Workshop\snippets\lab5\errandRequest.txt

__c. Note the code in the onMessage() method which reads an instance of an ErrandDataBean off the queue and then uses the CustomerController to make the errand request

__d. Add the necessary imports by right clicking in the editor and selecting **Source>Organize Imports**

__e. Format the code you just added by entering **Ctrl+Shift+F**

__f. Save the file by entering **Ctrl+S**

__3. To complete the Message Driven Bean we'll add an EJB references to the CustomerController EJB so its JNDI name won't have to be hard coded inside our message driven bean

__a. In the J2EE Navigator view, expand the **GoForItEJB** project and select **EJB Deployment Descriptor**. Double click on it to bring it up in its editor.

__b. Click on the **References** tab at the bottom of the editors window

__c. Under **References,** select **ErrandRequest** and click on **Add**

__d. Select **EJB local reference** and click on **Next** __e. Click on the **Browse** button next to **Link** and select the **CustomerController** EJB in the current project

__f. Click **OK**

__g. Click **Finish**

__h. Enter **Ctrl+S** to save your changes and then close the editor's window

__4. The code in the application that interfaces with the CustomerController uses an EJB references to it. You'll bind the reference in this step.

__a. In the J2EE Navigator view, expand the **GoForItWeb** project and select **Web Deployment Descriptor**. Double click on it to bring it up in its editor.

__b. Click on the **References** tab at the bottom of the editors window

__c. Select **EJB local** and click on **Add**

__d. Enter **ejb/CustomerController** as the name of the reference and click on the **Browse** button next to **Link** and select the **CustomerController** EJB.

__e. Click **OK**

__f. Enter **Ctrl+S** to save your changes and then close the editor's window

Part Five: Configuring the Embedded JMS Provider

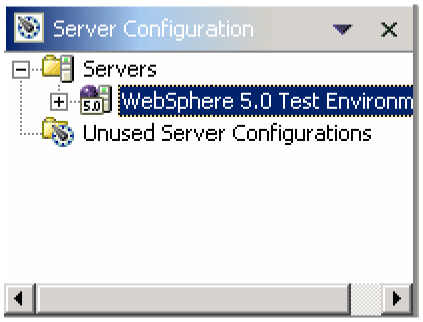
Prior to testing the Message Driven Bean, you'll have to configure the Embedded JMS Provider with a Queue and also create a listener within WebSphere that will listen for messages arriving on the Queue and invoke the Message Driven Bean when a message arrives.

__1. The first order of business is to configure the Embedded JMS Provider

__a. From the main menu in Application Developer, select **Window>Open Perspective>Other..**

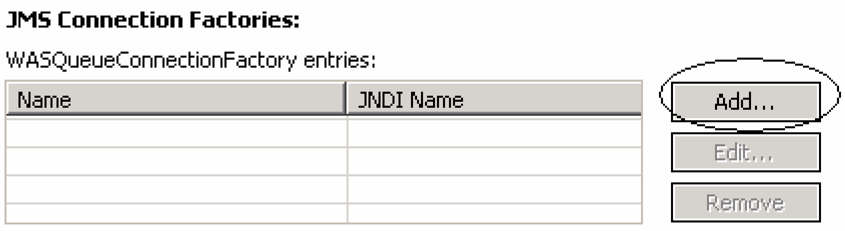
__b. Select **Server** from the pop up dialog and click on **OK**

__c. In the **Server Configuration** Window at the bottom left, expand **Servers** and double click on **WebSphere 5.0 Test Environment** to bring up the Server Configuration editor



__d. Select the **JMS** tab at the bottom of the editor's window

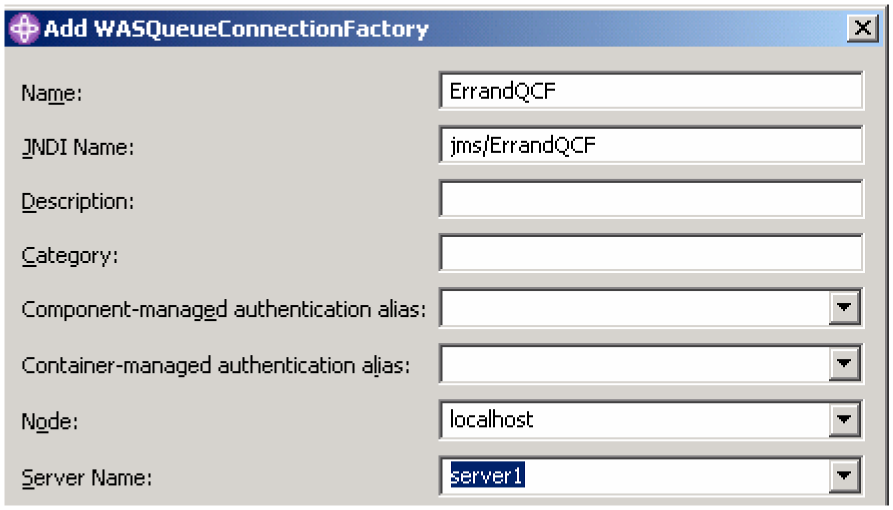
__e. Scroll down to the JMS Connection Factories section and click on Add next to WASQueueConnectionFactory entries


JMS Connection Factories:

WASQueueConnectionFactory entries:

Name	JNDI Name

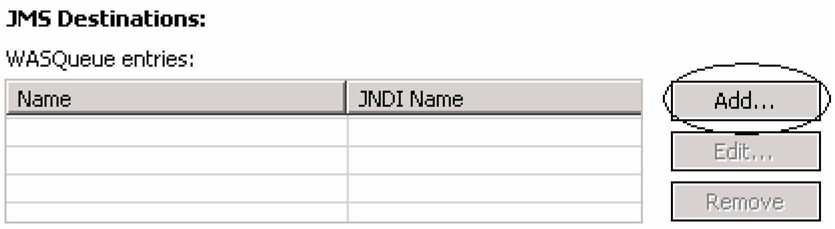
__f. Enter **ErrandQCF** as the **Name**, **jms/ErrandQCF** as the **JNDI Name**. Select **localhost** as the **Node** and **server1** as the **Server name**


Add WASQueueConnectionFactory

Name:	ErrandQCF
JNDI Name:	jms/ErrandQCF
Description:	
Category:	
Component-managed authentication alias:	
Container-managed authentication alias:	
Node:	localhost
Server Name:	server1

__g. Click on **OK**

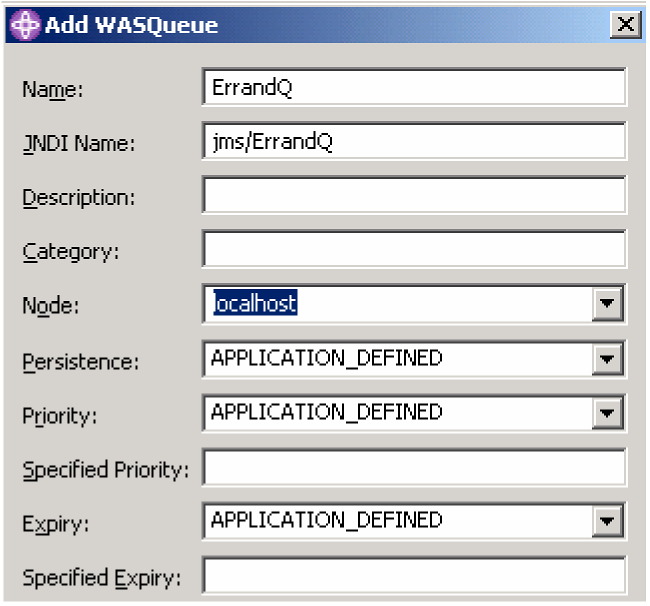
__h. Scroll down to the **JMS Destinations** section and click on **Add** next to **WASQueue entries**


JMS Destinations:

WASQueue entries:

Name	JNDI Name

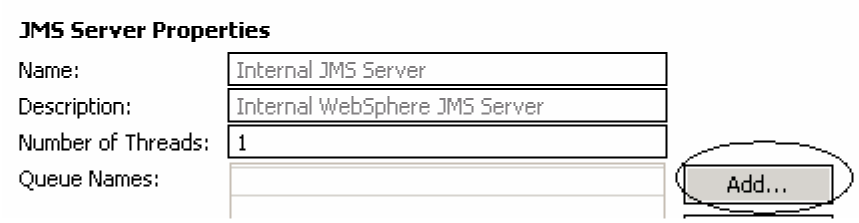
__i. Enter **ErrandQ** as the **Name**, **jms/ErrandQ** as the **JNDI Name**. Select **localhost** as the **Node**



Name:	ErrandQ
JNDI Name:	jms/ErrandQ
Description:	
Category:	
Node:	localhost
Persistence:	APPLICATION_DEFINED
Priority:	APPLICATION_DEFINED
Specified Priority:	
Expiry:	APPLICATION_DEFINED
Specified Expiry:	

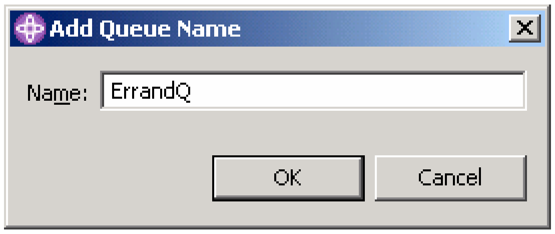
__j. Click **OK**

__k. Scroll up to the **JMS Server Properties** section and click on **Add** next to **Queue Names**



Name:	Internal JMS Server
Description:	Internal WebSphere JMS Server
Number of Threads:	1
Queue Names:	

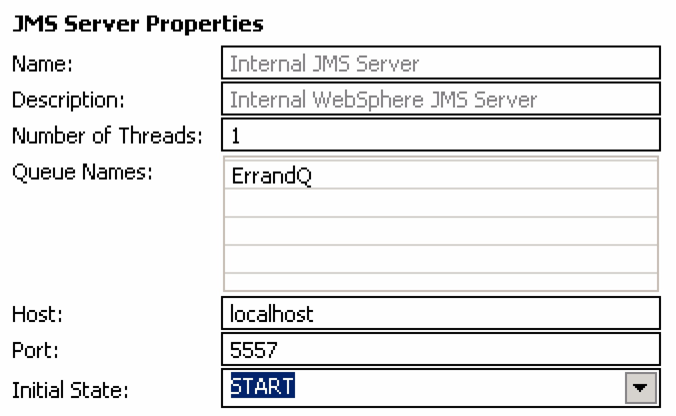
__l. Enter **ErrandQ** as the **Name**



Name: ErrandQ

__m. Click **OK**

__n. In the JMS Server Properties section, change the **Initial State** to **START**

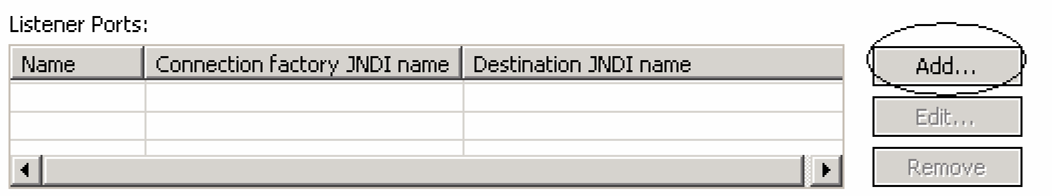

JMS Server Properties

Name:	Internal JMS Server
Description:	Internal WebSphere JMS Server
Number of Threads:	1
Queue Names:	ErrandQ
Host:	localhost
Port:	5557
Initial State:	START

__2. Next we'll configure the Listener

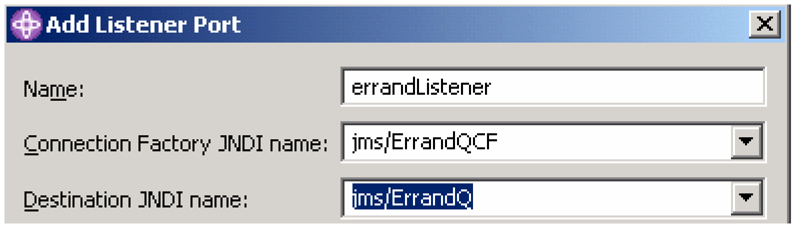
__a. Click on the **EJB** tab at the bottom of the editors window

__b. Click on the **Add** button next to the **Listener Ports** section


Listener Ports:

Name	Connection factory JNDI name	Destination JNDI name

__c. Enter **errandListener** as the **Name**. Select **jms/ErrandQCF** as the **Connection Factory JNDI Name** and **jms/ErrandQ** as the **Destination JNDI Name**


Add Listener Port

Name:	errandListener
Connection Factory JNDI name:	jms/ErrandQCF
Destination JNDI name:	jms/ErrandQ

__d. Click **OK**

__e. Enter **Ctrl+S** to save your changes

__f. Close the editors window

Part Six: Testing the Message Driven Bean

To test the Message Driven Bean you'll use a previously written J2EE client application that requests an errand by populating an instance of the ErrandDataBean class and puts it on the Queue that the Message Driven Bean listens to.

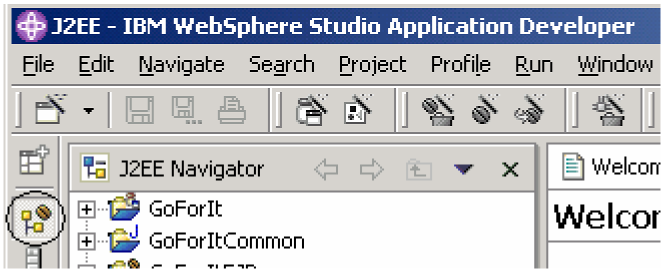
__1. The first order of business is to start the server

__a. Click on the **Servers** tab in the Tasks pane at the bottom right __b. Right click on the WebSphere 5.0 Test Environment server and select **Start**. Wait for the message in the console that says

Server server1 open for e-business before continuing

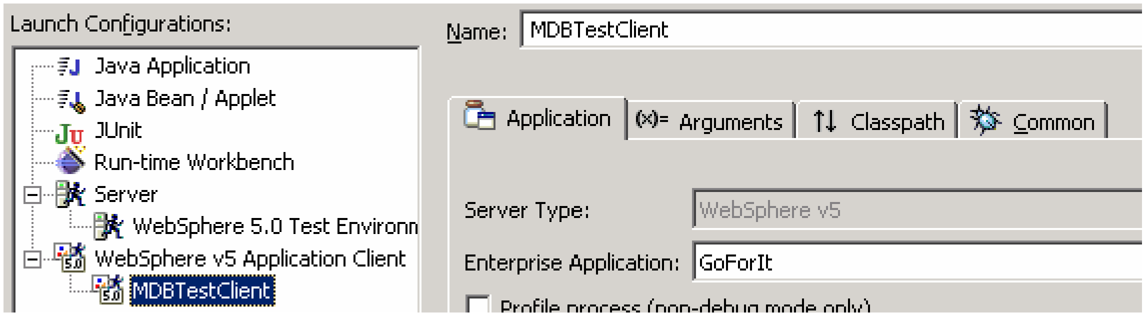
__2. Next you'll run the J2EE Client application

__a. Make sure you're in the J2EE Perspective by clicking on its icon in the left border of Application Developers window


J2EE - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Profile Run Window
J2EE Navigator
GoForIt
GoForItCommon
Welcom
Welcor

__b. From the main menu select **Run>Run..**

__c. In the **Launch Configurations** section, expand **WebSphere v5 Application Client** and select **MDBTestClient**



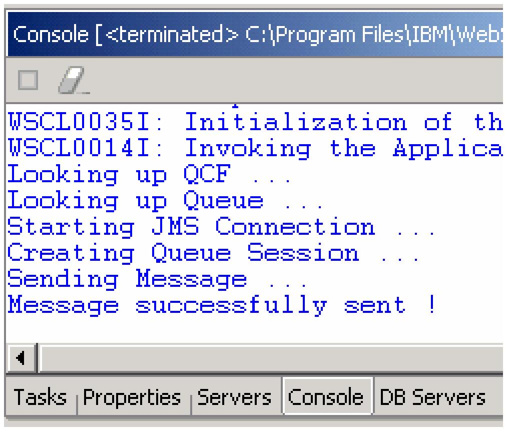
Name: MDBTestClient

Server Type: WebSphere v5

Enterprise Application: GoForIt

__d. Click **Run**

__e. The client application runs and its output appears in the console, verify that the last message is **Message successfully sent !**



```
WSCL0035I: Initialization of th
WSCL0014I: Invoking the Applica
Looking up QCF ...
Looking up Queue ...
Starting JMS Connection ...
Creating Queue Session ...
Sending Message ...
Message successfully sent !
```

__3. Finally you'll use the web application to verify that the Message Driven Bean worked __a. Start your browser and go to the URL **http://localhost:9080/GoForItWeb/index.jsp**.

__b. Login with userid **joeg** and a password of **password**.

__c. Click on the **View all errands** link

__d. The errand with the largest errand id should be the one created by the Message Driven Bean. Click on **View Details** for that errand and verify that the **Customer Comments** are **Requested via JMS Client**

__4. Stop the Server and exit Application Developer

__a. Click on the **Servers** tab in the Tasks pane at the bottom right

__b. Right click on the WebSphere 5.0 Test Environment server and select **Stop**

__c. Shutdown Application Developer (**File->Exit**)

What you did in this exercise

In this exercise you created and tested a Stateless Session Bean. You also created and tested a Message Driven Bean that processed errand requests that are sent as messages to WebSphere's Embedded JMS Provider.

IBM WebSphere Application Server v5.0 Workshop – LAB EXERCISE

Deploying EJBs to WebSphere Application Server v5.0

What this exercise is about

In this exercise, you will deploy the EJBs that you have thus far developed and tested within Application Developer. In the process you will work with the web based WebSphere 5.0 Admininstration Console.

What You Should Be Able To Do

You should be able to export from Application Developer , map the Entity beans to the previously created DataSource , configure the Embedded JMS Provider and deploy an EAR file to WebSphere with the WebSphere Admin Console.

Introduction

In this exercise you will export the Go-ForIt Enterprise Application from Application Developer as an EAR file and then start the WebSphere Admin Server and launch the Admin Console. Using the Admin Console, you will map the Entity beans to the DataSource that was created in the previous deployment lab and configure the Embedded JMS Provider. The exported EAR file will then be installed (deployed) to the Application server. Finally you'll start the application server and test the deployed application.

Exercise Instructions

Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__2. Start WebSphere Studio Application Developer

__d. From the Windows Start Menu select **Start>Programs-IBM WebSphere Studio>Application Developer**

__e. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab6**

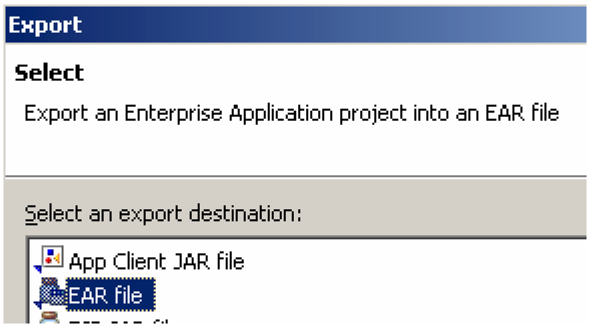
__f. Click **OK**

Part One: Exporting from Application Developer

__1. Run the EAR export wizard to export the Go-ForIt application

__a. Select **File->Export** from the main menu

__b. Select **EAR file** as the export destination



Export

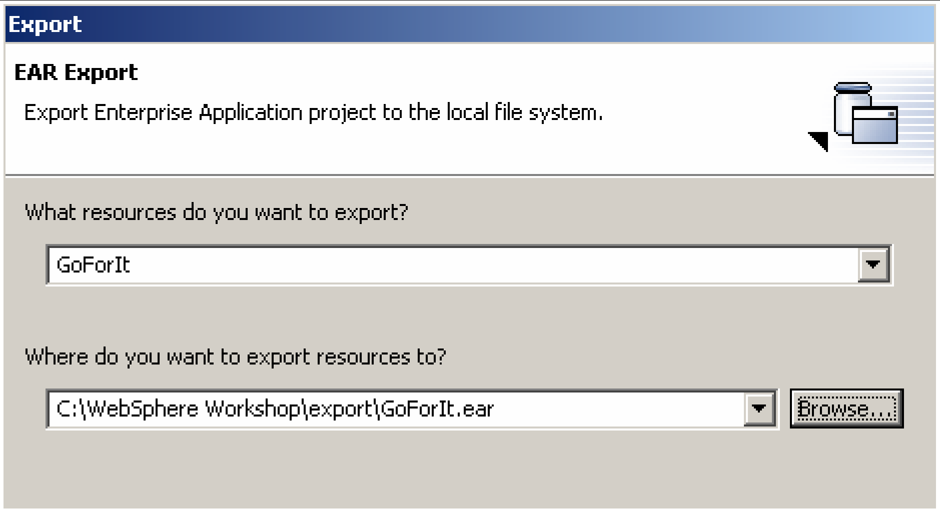
Select

Export an Enterprise Application project into an EAR file

Select an export destination:

__c. Click **Next**

__d. Select **GoForIt** as the resource to export and save it as **GoForIt.ear** in the **C:\WebSphere Workshop\export** folder.



EAR Export

Export Enterprise Application project to the local file system.

What resources do you want to export?

GoForIt

Where do you want to export resources to?

C:\WebSphere Workshop\export\GoForIt.ear

__e. Click **Finish**

__f. Click **Yes** to overwrite the GoForIt.ear file from the previous deployment lab

__2. Exit from Application Developer

__b. **File->Exit** from the main menu

Part Two: Configuring WebSphere and Deploying the Enterprise Application

__1. Start the WebSphere Application Server

__a. Open a Command Prompt and change to the **C:\Program Files\WebSphere\AppServer\bin** directory

__b. Enter **startserver server1** at the command prompt

__c. The server will be fully started when the process ID is listed as shown below (Note: your process ID will most likely be different from the one shown below)

```
C:\Program Files\WebSphere\AppServer\bin>startserver server1
ADMU0116I: Tool information is being logged in file C:\Program
           Files\WebSphere\AppServer\logs\server1\startServer.log
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 1772
```

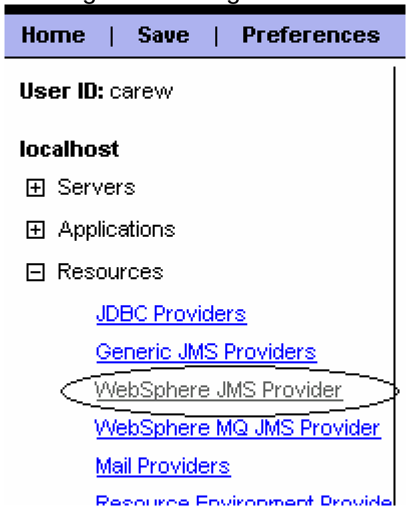
__2. Add the QueueConnectionFactory and Queue into the JNDI namespace

__a. Start the WebSphere Admin Console by selecting the following from the Windows Start Menu

Start>Programs>IBM WebSphere Application Server v5.0>Administrative Console

__b. The console will open in your browser with a prompt for a user ID. Enter any ID and click on **OK**.

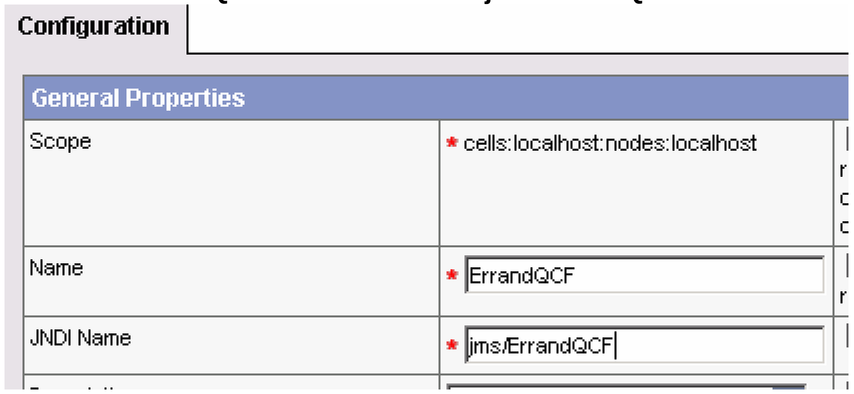
__c. Expand the **Resources** tree on the left and click **WebSphere JMS Provider** to begin the configuration of the JMS Provider



__d. In the Work Area frame scroll down to the **Additional Properties** section and click on **WebSphere Queue Connection Factories**

__e. Click on **New**

__f. Enter **ErrandQCF** as the **Name** and **jms/ErrandQCF** as the **JNDI Name**



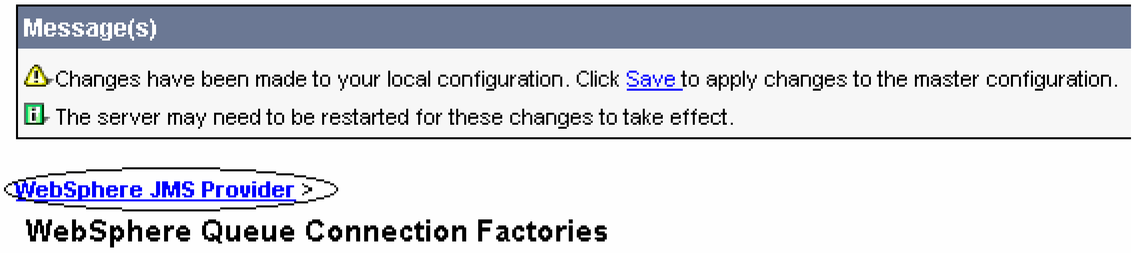
Configuration

General Properties

Scope	* cells:localhost:nodes:localhost
Name	* ErrandQCF
JNDI Name	* jms/ErrandQCF

__g. Click **OK**

__h. Click on the **WebSphere JMS Provider** link at the top of the Work area



Message(s)

Changes have been made to your local configuration. Click Save to apply changes to the master configuration.

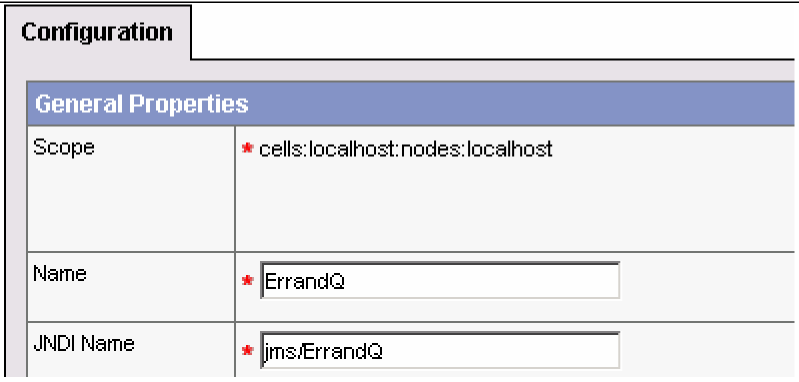
The server may need to be restarted for these changes to take effect.

WebSphere JMS Provider >

WebSphere Queue Connection Factories

__i. Click on **WebSphere Queue Destinations** in the **Additional Properties** section

__j. Enter **ErrandQ** as the **Name** and **jms/ErrandQ** as the **JNDI Name**



Configuration

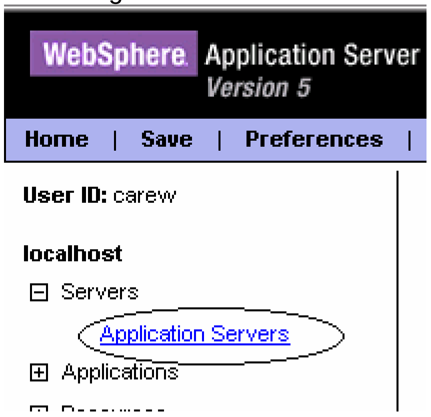
General Properties

Scope	* cells:localhost:nodes:localhost
Name	* ErrandQ
JNDI Name	* jms/ErrandQ

__k. Click **OK**

__3. Configure the Embedded JMS Provider

__a. Expand the **Servers** tree on the left and click **Application Servers** to begin the configuration of the JMS Provider

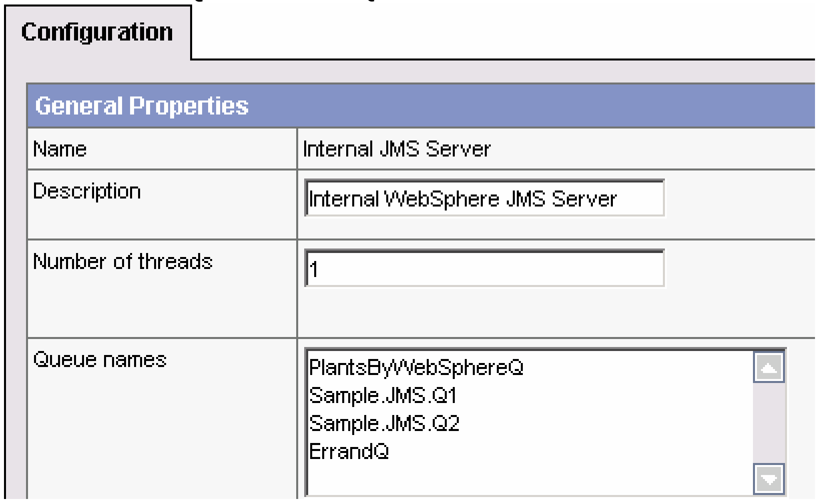

WebSphere Application Server Version 5
Home | Save | Preferences |
User ID: carew
localhost
Servers
Application Servers
Applications

__b. In the Work Area frame click on **server1**

__c. Scroll down to the **Additional Properties** section and click on **Server Components**

__d. Click on **JMS Servers**

__e. Add **ErrandQ** to the list of Queue Names



Configuration

General Properties

Name	Internal JMS Server
Description	Internal WebSphere JMS Server
Number of threads	1
Queue names	PlantsByWebSphereQ Sample.JMS.Q1 Sample.JMS.Q2 ErrandQ

__f. Click OK

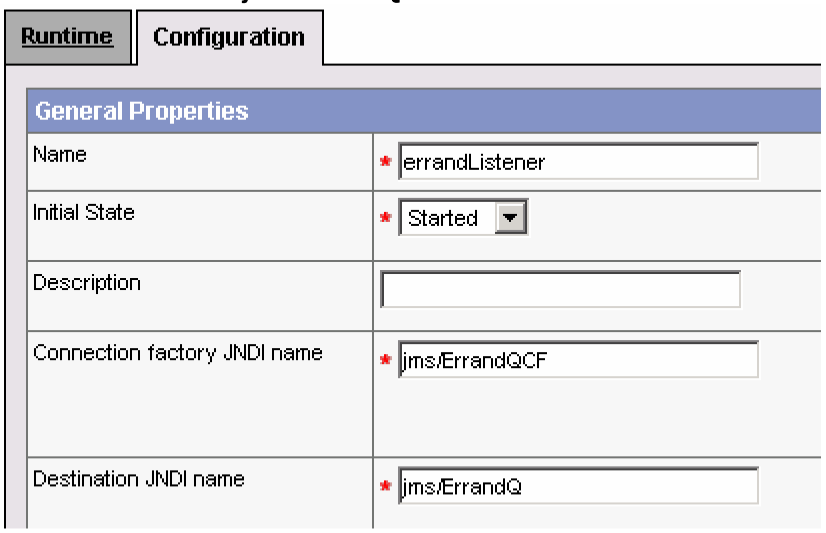
__4. Configure the Message Listener

__**a.** Scroll down to the **Additional Properties** section and click on **Message Listener Service**

__b. Click **Listener Ports**

__c. Click **New**

__d. Enter **errandListener** as the **Name**, **jms/ErrandQCF** as the **Connection factory JNDI Name** and **jms/ErrandQ** as the **Destination JNDI name**



General Properties

Name	* errandListener
Initial State	* Started
Description	
Connection factory JNDI name	* jms/ErrandQCF
Destination JNDI name	* jms/ErrandQ

__e. Click **OK**

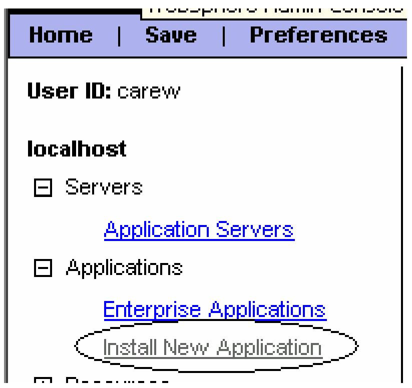
__5. Save your changes

__a. Click **Save** from the header at the upper left of the page

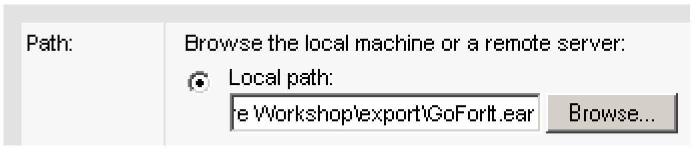
__b. The Save page will be displayed. Click **Save**

__6. Install the Enterprise Application

__a. Expand the **Applications** tree and click on **Install New Application**



__b. Click **Browse** for the local path and navigate to **C:\WebSphere Workshop\export\GoForIt.ear**



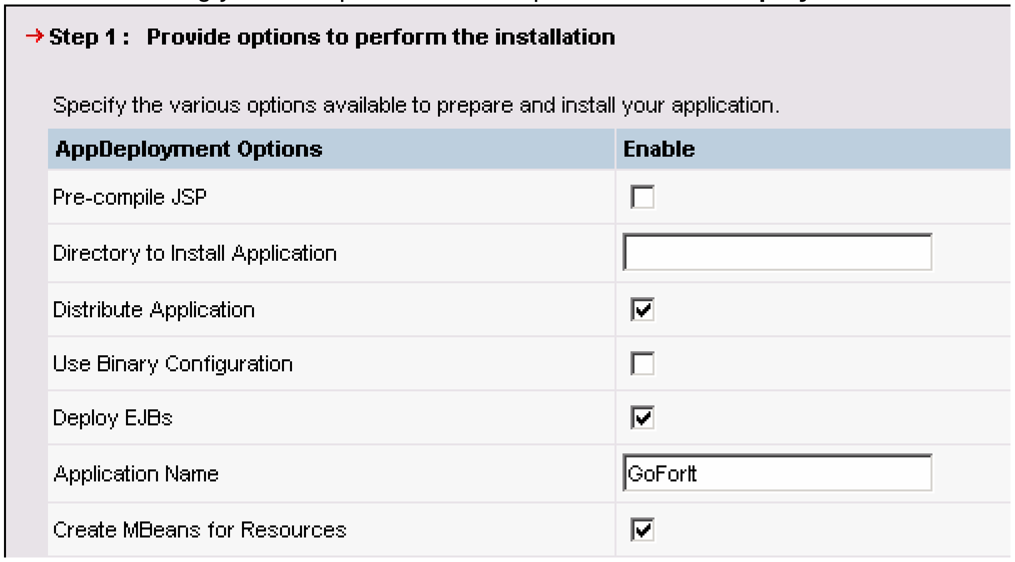
Path: Browse the local machine or a remote server:

(●) Local path: e Workshop\export\GoForIt.ear

__c. Click **Next**

__d. You can accept the default bindings (specified in the .ear file) by clicking **Next** again

__e. This will bring you to Step 1 of the install process. Check **Deploy EJBs**



→ Step 1 : Provide options to perform the installation

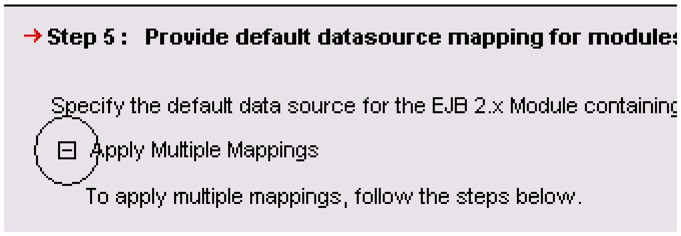
Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	☐
Directory to Install Application	
Distribute Application	☑
Use Binary Configuration	☐
Deploy EJBs	☑
Application Name	GoForIt
Create MBeans for Resources	☑

__f. Click **Next**

__g. Accept the defaults for **Step 2** and click on **Step 5**

__h. Expand **Apply Multiple Mappings**



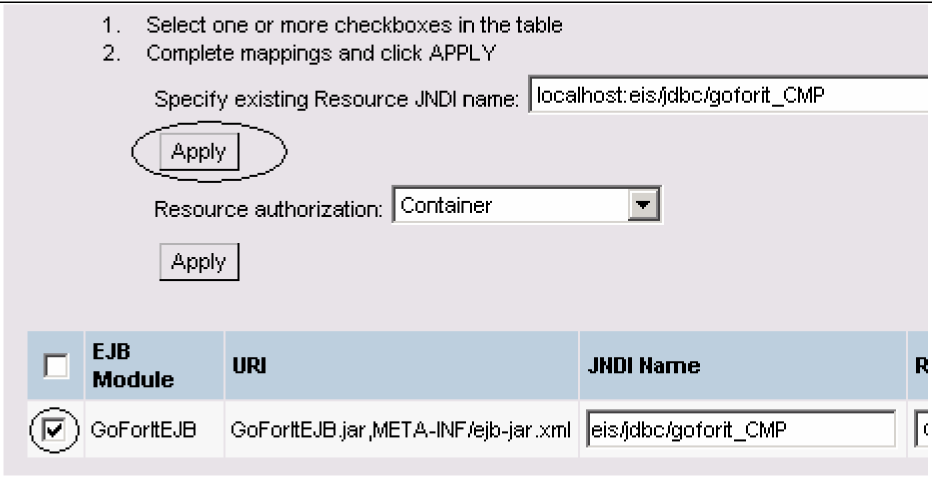
→ Step 5 : Provide default datasource mapping for module

Specify the default data source for the EJB 2.x Module containing

⊟ Apply Multiple Mappings

To apply multiple mappings, follow the steps below.

__i. Select **localhost:eis/jdbc/goforit_CMP** for the **Resource JNDI name** *AND* check the checkbox next to **GoForItEJB** and then click **Apply**



1. Select one or more checkboxes in the table
2. Complete mappings and click APPLY

Specify existing Resource JNDI name: localhost:eis/jdbc/goforit_CMP

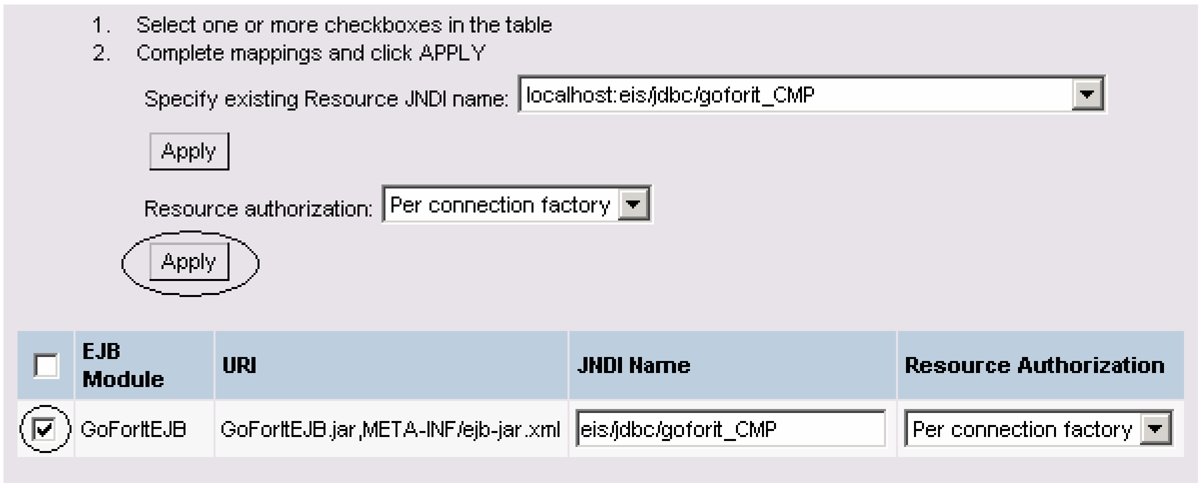
Apply

Resource authorization: Container

Apply

	EJB Module	URI	JNDI Name	R
☐				
☑	GoForItEJB	GoForItEJB.jar,META-INF/ejb-jar.xml	eis/jdbc/goforit_CMP	

__j. Select **Per connection factory** as the **Resource authorization** *AND* check the checkbox next to **GoForItEJB** and thenclick **Apply**



1. Select one or more checkboxes in the table
2. Complete mappings and click APPLY

Specify existing Resource JNDI name: localhost:eis/jdbc/goforit_CMP

Apply

Resource authorization: Per connection factory

Apply

	EJB Module	URI	JNDI Name	Resource Authorization
☐				
☑	GoForItEJB	GoForItEJB.jar,META-INF/ejb-jar.xml	eis/jdbc/goforit_CMP	Per connection factory

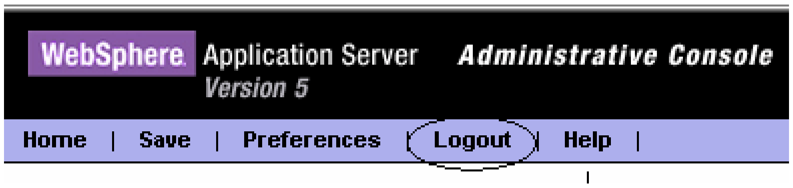
__k. Click on **Step 12** and then click **Finish**

__l. The EJB Deploy code will be regenerated. Wait for the message that says that the application was successfully installed

__m. Save the changes by clicking on **Save to Master Configuration** and then **Save** in the Save Page

__7. Exit the Admin Console

__a. Exit the Admin Console by clicking on **Logout**


WebSphere Application Server Version 5
Administrative Console
Home | Save | Preferences | Logout | Help |

__b. Shutdown down your browser

__8. Regenerate the plugin configuration for the IBM HTTP Server and restart the server to realize the configuration changes

__d. Go back to the command window that you used to start the server and type

genplugincfg

__e. When the command completes, stop the server by entering the following command

stopserver server1

__f. When the stopserver commands completes, start the server again by entering the following command

startserver server1

Part Three: Testing the Application

__1. Test the application

__a. Start your browser and go to the URL **http://localhost/GoForItWeb/index.jsp** . Verify that the login page comes up.

__b. Login with userid **joeg** and a password of **password**. Verify that the main menu page comes up

__c. Click on **View All Errands** and note the largest errand id in the list

__d. Go back to the command prompt that you used to start the server and enter the following command to run the J2EE client application that we use to test the Message Driven Bean

launchClient "C:\Program Files\WebSphere\AppServer\installedApps\localhost\GoForIt.ear"

__e. Refresh the page in your browser that has the list of errands and verify that a new errand has been added to the list

__2. Stop the Server

__a. Go back to the command window that you used to start the server and type

stopserver server1

What you did in this exercise

In this lab you learned how to deploy EJBs to the WebSphere 5.0 Application Server. You configured the mapping between the Entity Beans and a Data Source and you configured the Embedded JMS Provider and a Message Listener for the Message Driven Bean. You then installed the GoForIt application and tested it.

IBM WebSphere Application Server v5.0 Workshop – LAB EXERCISE

Adding a Web Services Interface to the GoForIt Application

What this exercise is about

In this exercise, you will expose the errand request functionality of the GoForIt Application as a Web Service (SOAP over HTTP) using the Web Services support in Application Developer.

What You Should Be Able To Do

You should be able to deploy existing parts of your Enterprise applications as Web Services and test those Web Services using the tooling in Application Developer.

Introduction

In this exercise you will use the Web Services wizard in Application Developer to deploy the requestErrand() method of the Business Delegate interface to the CustomerController Session Bean as a SOAP service using the Web Services wizard in Application Developer. You will then run a generated SOAP test client to test the creation of an errand request. You will also monitor the SOAP traffic over TCP/IP to see how the SOAP requests and responses are actually sent using HTTP.

Exercise Instructions

Setup:

In this step you'll start WebSphere Studio Application Developer with the workspace containing the files for this lab.

__1. Start WebSphere Studio Application Developer

__a. From the Windows Start Menu select **Start>Programs-IBM WebSphere Studio>Application Developer**

__b. A dialog box will be displayed allowing you to select the location of the Workspace. Click the **Browse…** button and browse to **C:\WebSphere Workshop\wsad\workspaces\Lab7**

__c. Click **OK**

Part One: Generating the Web Service

__1. Run the Web Service wizard

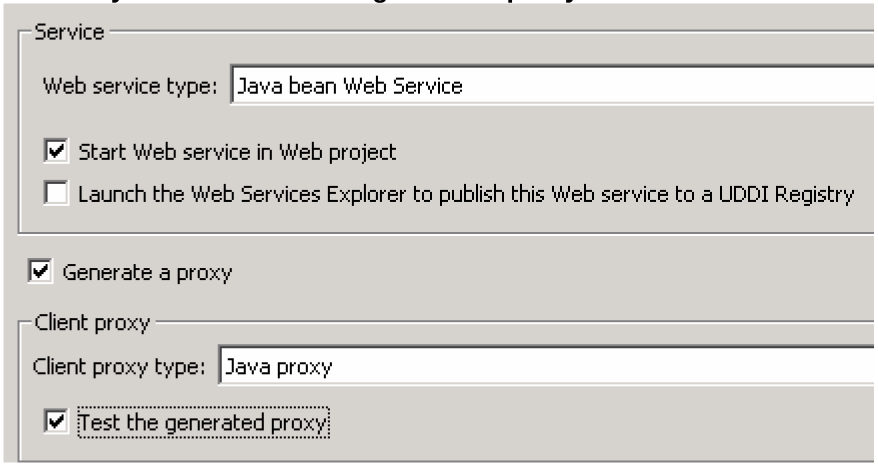
__a. In the J2EE Navigator view of the J2EE Perspective, select the **GoForItWeb** project

__b. Right click and select **New>Other**

__c. In the pane of the left, select **Web Services** and in the pane on the right select **Web Service**

__d. Click **Next**

__e. Select **Java Bean Web Service** as the **Web service type**, check **Generate a Proxy** and check **Test the generated proxy**



Service

Web service type: Java bean Web Service

☑ Start Web service in Web project

☐ Launch the Web Services Explorer to publish this Web service to a UDDI Registry

☑ Generate a proxy

Client proxy

Client proxy type: Java proxy

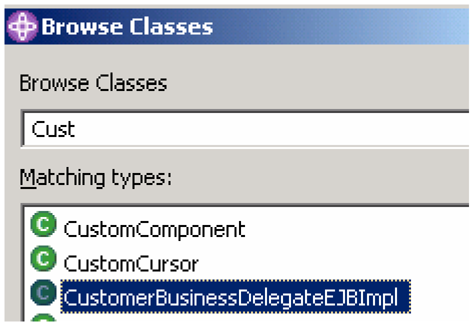
☑ Test the generated proxy

__f. Click **Next**

__g. Accept the defaults for the deployment settings and click **Next**

__h. Click on **Browse classes next** to the **Bean** field

__i. Enter **Cust** to narrow the search and then select **CustomerBusinessDelegateEJBImpl**



Browse Classes

Browse Classes

Cust

Matching types:

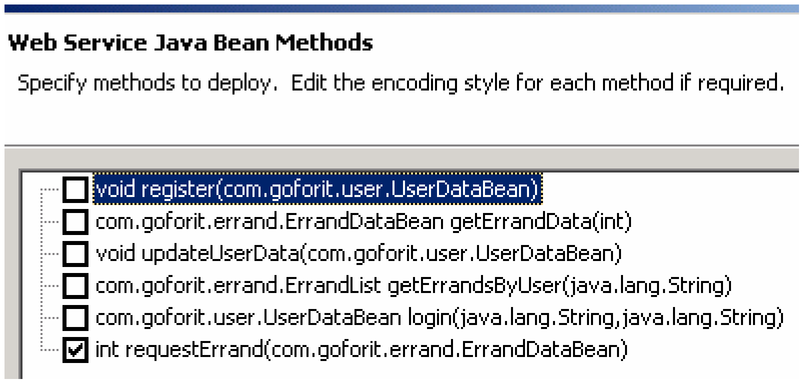
CustomComponent
CustomCursor
CustomerBusinessDelegateEJBImpl

__j. Click **OK**

__k. Click **Next**

__l. Change the Web Service URI to http://www.goforit.com/com.goforit.bd.CustomerBusinessDelegateEJBImpl and click **Next**

__m. Click **Deselect All** and then select the method **requestErrand(com.goforit.errand.ErrandDataBean)**



Web Service Java Bean Methods

Specify methods to deploy. Edit the encoding style for each method if required.

- [ ] void register(com.goforit.user.UserDataBean)
- [ ] com.goforit.errand.ErrandDataBean getErrandData(int)
- [ ] void updateUserData(com.goforit.user.UserDataBean)
- [ ] com.goforit.errand.ErrandList getErrandsByUser(java.lang.String)
- [ ] com.goforit.user.UserDataBean login(java.lang.String,java.lang.String)
- [✓] int requestErrand(com.goforit.errand.ErrandDataBean)

__n. Click **Next**

__o. Accept the defaults for the Web Service Test properties and click **Next**

__p. Accept the defaults for the Web Service Binding Proxy Generation and click **Next**

__q. Accept the defaults for the Web Service Test properties and click **Next**

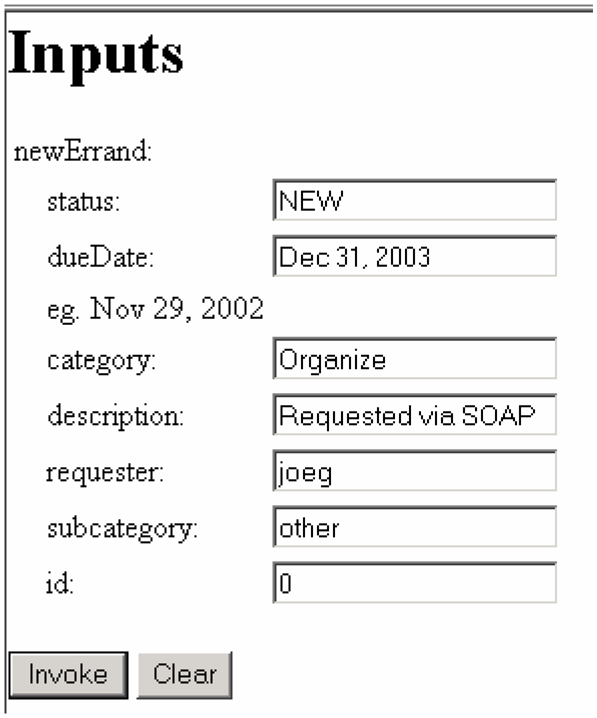
__r. Accept the defaults for the Publishing options and click **Finish.** This will generate the Web Service and add the Apache SOAP server to the GoForItWeb project.

Finally it will launch your browser with the generated test client for the Web Service.

__2. Test the Web Service

__a. In the **Methods** frame in the browser click on **requestErrand**

__b. In the **Inputs** Frame, enter the values shown below



Inputs

newErrand:

status:	NEW
dueDate:	Dec 31, 2003
eg. Nov 29, 2002	
category:	Organize
description:	Requested via SOAP
requester:	joeg
subcategory:	other
id:	0

__c. Click **Invoke**

__d. Verify that an Errand id and not an error is returned in the **Results** frame. Keep the browser open for the next part of the lab

<u>Part Two: Monitoring the SOAP requests</u>

To monitor the SOAP request you'll use a feature of Application Developer called the TCP/IP Monitoring Server that is equivalent in functionality to the TcpTunnel tool that is packaged with Apache SOAP. Both tools allow the monitoring of TCP/IP network traffic, not only helping you debug your SOAP application, but also providing help in learning more about the SOAP messages that flow back and forth in your system.

A TCP/IP Monitoring Server will listen to TCP/IP messages arriving on a particular port, display them in a window and then forward the message to its ultimate destination. The same happens with any returned result.

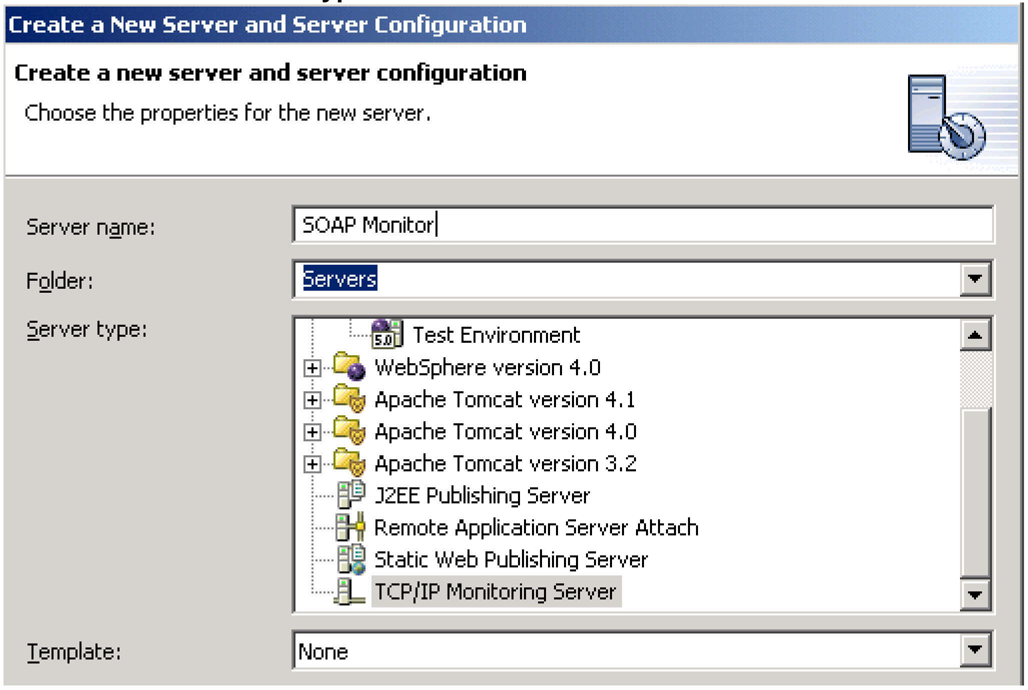
__1. Configure the TCP/IP Monitoring Server

__a. From the main menu select **File>New>Other**

__b. Select **Server** in the pane on the left and **Server and Server Configuration** in the pane on the right

__c. Click Next

__d. Enter **SOAP Monitor** as the **Server Name** and select **TCP/IP Monitoring Server** as the **Server type**


Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: SOAP Monitor

Folder: Servers

Server type:
- Test Environment
- WebSphere version 4.0
- Apache Tomcat version 4.1
- Apache Tomcat version 4.0
- Apache Tomcat version 3.2
- J2EE Publishing Server
- Remote Application Server Attach
- Static Web Publishing Server
- TCP/IP Monitoring Server

Template: None

__e. Click **Finish**

__f. In the Servers Pane , select the **SOAP Monitor** server, right click and select **Start**

__g. There should be a message in the console saying that the server is listening on Port 9081 and forwarding to port 9080

__2. Test the Web Service again to see the SOAP network traffic. We'll have to use port 9081 for the SOAP request in order to be able to monitor it.

__a. Go back to the browser window that was running the test client and click on **getEndPoint** in the **Methods** frame

__b. Click on **Invoke** in the **Inputs** frame

__c. Highlight the URL returned in the **Results** frame and enter **Ctrl+C**

__d. Click on **setEndPoint** in the **Methods** frame

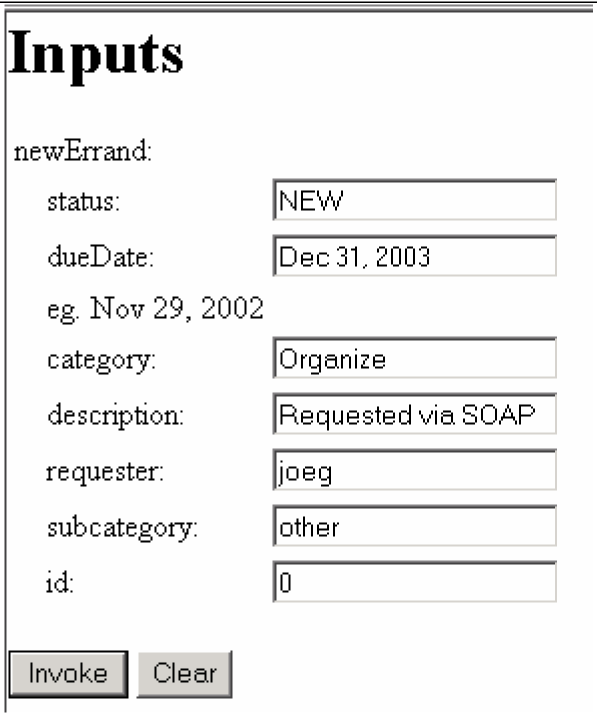
__e. Put your cursor in the **url** text field (in the **Inputs** frame) and enter **Ctrl+v**

__f. Change the port number of the URL (leave the rest of it unchanged) in the text field to **9081**

__g. Click **Invoke**

__h. Click on **requestErrand** in the **Methods** frame

__i. Enter the following in the **Inputs** frame



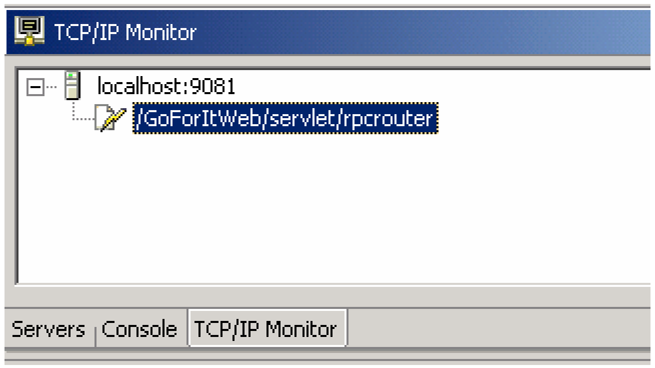
Inputs

newErrand:

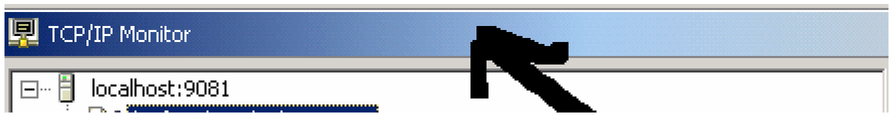
status:	NEW
dueDate:	Dec 31, 2003
eg. Nov 29, 2002	
category:	Organize
description:	Requested via SOAP
requester:	joeg
subcategory:	other
id:	0

__j. Click on **Invoke**

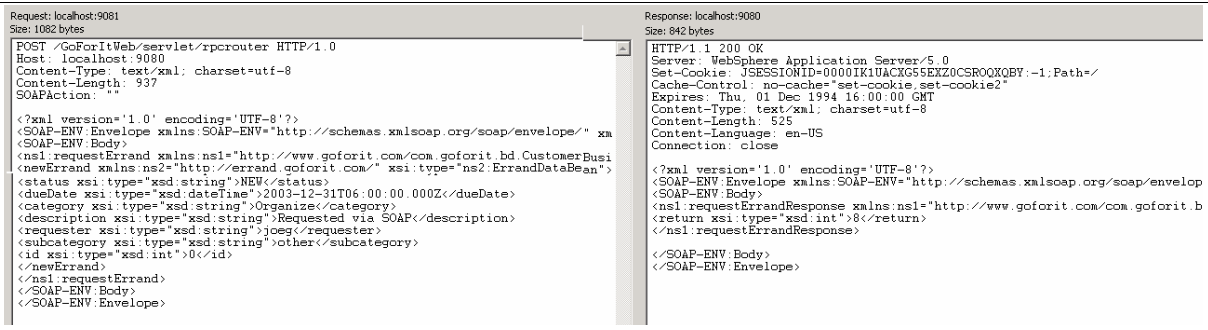
__k. Go back to Application Developer expand **localhost:9081** and select **/GoForItWeb/servlet/rpcrouter** in the **TCP/IP Monitor** window



__l. Double click on the top border of the window to make it larger so the SOAP request and response is visible



__m. The SOAP request and response should now be visible



Request: localhost:9081
Size: 1082 bytes

```
POST /GoForItWeb/servlet/rpcrouter HTTP/1.0
Host: localhost:9080
Content-Type: text/xml; charset=utf-8
Content-Length: 937
SOAPAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xm
<SOAP-ENV:Body>
<ns1:requestErrand xmlns:ns1="http://www.goforit.com/com.goforit.bd.CustomerBusi
<newErrand xmlns:ns2="http://errand.goforit.com/" xsi:type="ns2:ErrandDataBean">
<status xsi:type="xsd:string">NEW</status>
<dueDate xsi:type="xsd:dateTime">2003-12-31T06:00:00.000Z</dueDate>
<category xsi:type="xsd:string">Organize</category>
<description xsi:type="xsd:string">Requested via SOAP</description>
<requester xsi:type="xsd:string">joeg</requester>
<subcategory xsi:type="xsd:string">other</subcategory>
<id xsi:type="xsd:int">0</id>
</newErrand>
</ns1:requestErrand>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response: localhost:9080
Size: 842 bytes

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Set-Cookie: JSESSIONID=0000IK1UACXG55EXZ0CSROQXQBY:-1;Path=/
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 525
Content-Language: en-US
Connection: close

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelop
<SOAP-ENV:Body>
<ns1:requestErrandResponse xmlns:ns1="http://www.goforit.com/com.goforit.b
<return xsi:type="xsd:int">8</return>
</ns1:requestErrandResponse>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

What you did in this exercise

In this exercise you used the Web Services wizard in Application Developer to expose the errand request functionality of the GoForIt application as a SOAP service and accessed it using a test client generated by the wizard. You also used the TCP/IP Monitoring Server in Application Developer to monitor the SOAP request from the test client and the response to it.