IBM WebSphere® Application Server Feature Pack for Web 2.0 – LAB EXERCISE

Creating a Hello World application

What this exercise is about

In this exercise you are going to create a simple Hello World Application using Rational Application Developer V7.5 and the Feature Pack for Web 2.0.

Lab requirements

For this lab your system must have Rational Application Developer 7.5 installed and an associated WebSphere Application Server 6.1 test environment.

What you should be able to do

At the end of this lab you should be able to:

- Understand how to using the Rational Application Developer Web 2.0 features
- Know how to create a basic Web 2.0 application

Introduction

This lab shows you how to create a simple Hello World application. The application simply has a name textbox and a button, and when you click on the button an alert box pops up with the given name. To create the application you use Rational Application Developer V7.5 and create a Dynamic Web Project and bring in Dojo and the Feature Pack for Web 2.0 Server-side technologies. You use the drag-and-drop Dojo palette within Rational Application Developer to create the client side code of a textbox and a button. You then create a servlet and edit the doPost method.

Exercise instructions

Instructions and subsequent documentation use symbolic references to directories which are listed as follows:

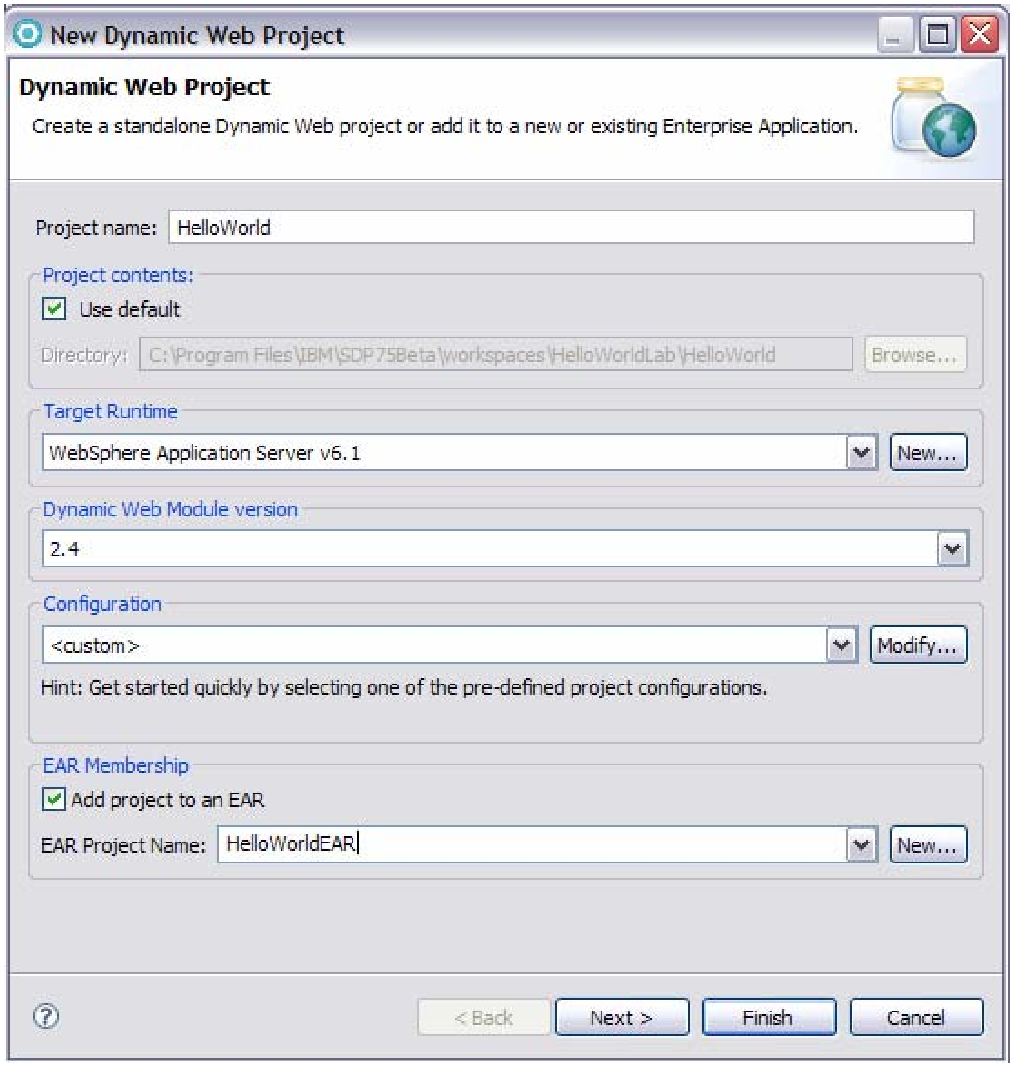
Reference Variable	Windows Location
<WAS_HOME>	C:\Program Files\IBM\SDP75\runtimes\base_V61

Part 1: Create a new application

____ 1. Start the Rational Application Developer. Go to **Start > All Programs > IBM Software Delivery Platform > IBM Rational Application Developer 7.5 > IBM Rational Application Developer**

____ 2. Create a new application, go to **File > New > Dynamic Web Project**

__ a. Enter **HelloWorld** for the Project name. Choose the Target Runtime as WebSphere **Application Server V6.1**. For EAR Membership **checkmark** the Add Project to and EAR check box and enter EAR Project Name of **HelloWorldEAR** as in the picture below:


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Browse...
New...
Modify...
< Back
Next >
Finish
Cancel

Project name: HelloWorld

Project contents:

[✓] Use default

Directory: C:\Program Files\IBM\SDP75Beta\workspaces\HelloWorldLab\HelloWorld

Target Runtime: WebSphere Application Server v6.1

Dynamic Web Module version: 2.4

Configuration: <custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

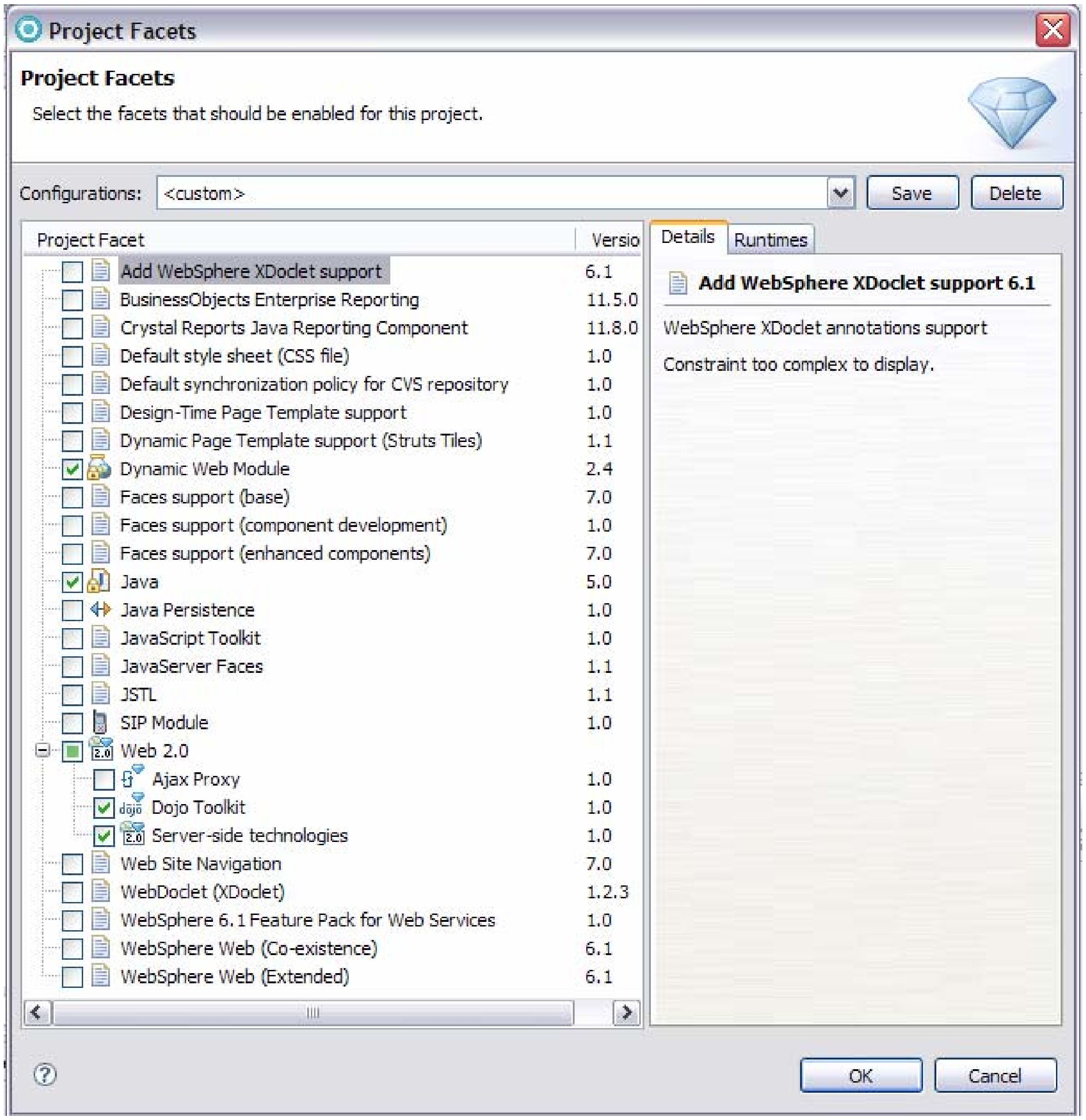
EAR Membership

[✓] Add project to an EAR

EAR Project Name: HelloWorldEAR

__ b. Select the **Modify…** button in the Configuration section

__ c. Expand the **Web 2.0** section and select **Dojo Toolkit** and **Server-side technologies**



Project Facets

Select the facets that should be enabled for this project.

Configurations: <custom>

Project Facet	Version
☐ Add WebSphere XDoclet support	6.1
☐ BusinessObjects Enterprise Reporting	11.5.0
☐ Crystal Reports Java Reporting Component	11.8.0
☐ Default style sheet (CSS file)	1.0
☐ Default synchronization policy for CVS repository	1.0
☐ Design-Time Page Template support	1.0
☐ Dynamic Page Template support (Struts Tiles)	1.1
✓ Dynamic Web Module	2.4
☐ Faces support (base)	7.0
☐ Faces support (component development)	1.0
☐ Faces support (enhanced components)	7.0
✓ Java	5.0
☐ Java Persistence	1.0
☐ JavaScript Toolkit	1.0
☐ JavaServer Faces	1.1
☐ JSTL	1.1
☐ SIP Module	1.0
■ Web 2.0	
☐ Ajax Proxy	1.0
✓ Dojo Toolkit	1.0
✓ Server-side technologies	1.0
☐ Web Site Navigation	7.0
☐ WebDoclet (XDoclet)	1.2.3
☐ WebSphere 6.1 Feature Pack for Web Services	1.0
☐ WebSphere Web (Co-existence)	6.1
☐ WebSphere Web (Extended)	6.1

Details

Add WebSphere XDoclet support 6.1

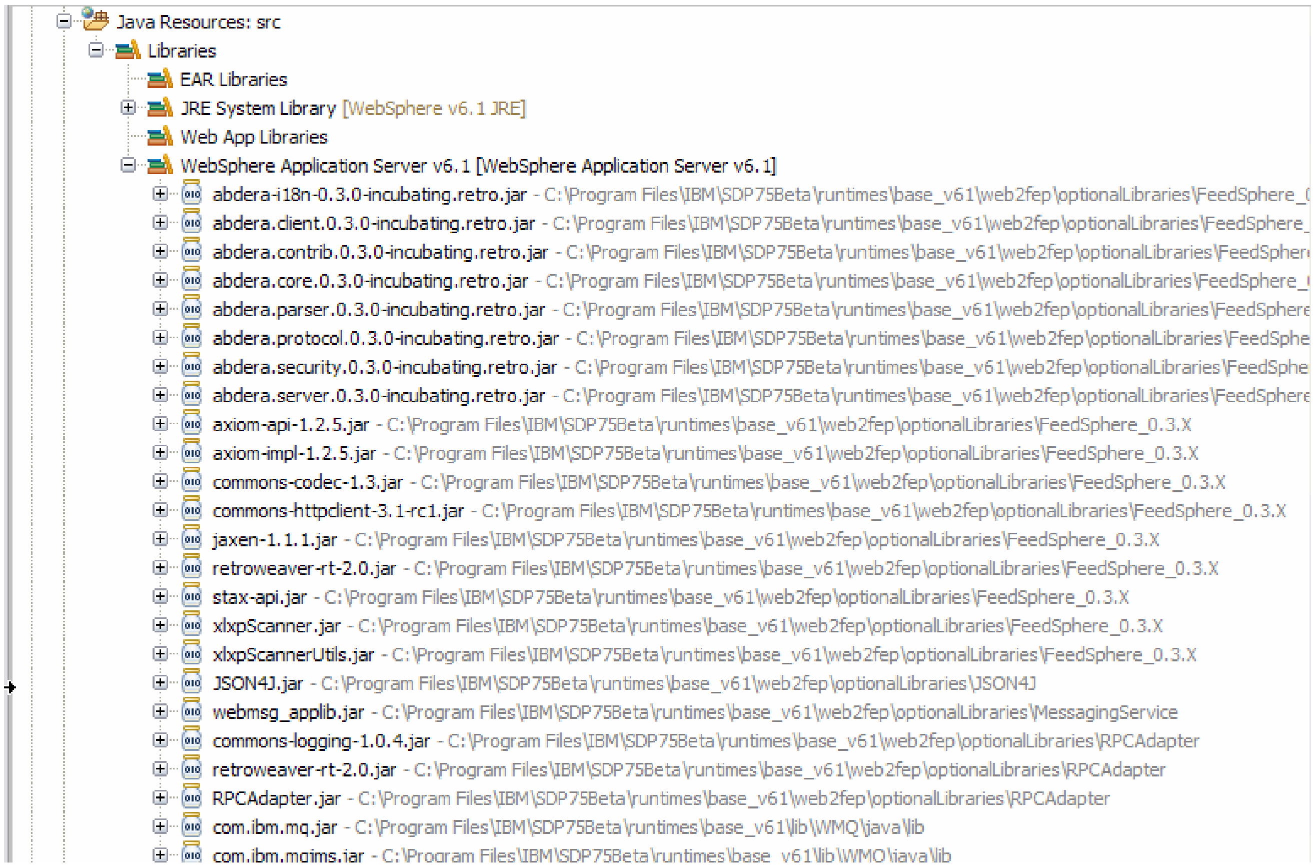
WebSphere XDoclet annotations support

Constraint too complex to display.

__ d. Select **Next > Next > Finish** to create your application

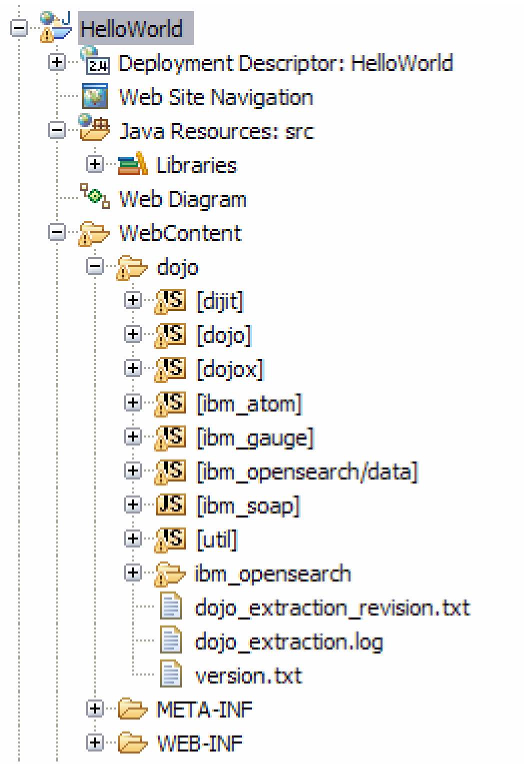
__ e. Your application will be created and Rational Application Developer will go and validate your new application. Note: you will see a lot of Warnings. These are warnings that are within Dojo, you can ignore these warnings.

____ 3. Walk through your new application. First expand out **Java Resources: src > Libraries > WebSphere Application server V6.1** and notice the jar files that are included from the web2fep folder. These are included since you choose to include the Server-side technologies when you created your application.



```
Java Resources: src
  Libraries
    EAR Libraries
    JRE System Library [WebSphere v6.1 JRE]
    Web App Libraries
    WebSphere Application Server v6.1 [WebSphere Application Server v6.1]
      abdera-i18n-0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_
      abdera.client.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_
      abdera.contrib.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSpher
      abdera.core.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_
      abdera.parser.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere
      abdera.protocol.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphe
      abdera.security.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphe
      abdera.server.0.3.0-incubating.retro.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere
      axiom-api-1.2.5.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      axiom-impl-1.2.5.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      commons-codec-1.3.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      commons-httpclient-3.1-rc1.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      jaxen-1.1.1.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      retroweaver-rt-2.0.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      stax-api.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      xlxpScanner.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      xlxpScannerUtils.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\FeedSphere_0.3.X
      JSON4J.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\JSON4J
      webmsg_applib.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\MessagingService
      commons-logging-1.0.4.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\RPCAdapter
      retroweaver-rt-2.0.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\RPCAdapter
      RPCAdapter.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\web2fep\optionalLibraries\RPCAdapter
      com.ibm.mq.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\lib\WMQ\java\lib
      com.ibm.mqjms.jar - C:\Program Files\IBM\SDP75Beta\runtimes\base_v61\lib\WMQ\java\lib
```

____ 4. Expand the Web Content folder and then the dojo folder. Notice all the dojo imports and the IBM extensions to the Dojo Toolkit (ibm_atom, ibm_gauge, ibm_opensearch, ibm_soap).



__ a. Take a look at some of the dojo widgets to familiarize yourself with the capabilities of Dojo.

1) Select **dojo > dijit > themes** then right click **themeTester.html** and choose **Open with > Web Browser**. This html page shows a lot of the widgets that are available to use with Dojo. To view the code that is used to bring in these widgets into your Web page right click **themeTester.html** and choose **Open with > HTML Editor**. This test page demonstrates many of the dijit widgets and their capabilities. It offers a quick look at how the widgets interact and how themes, such as Tundra, can provide polished appearance to dojo.

2) Select **dojo > dojox > grid >** tests then right click **test_edit_dijit.html** and choose **Open with > Web Browser**. This html page shows the grid widget that is available to use with Dojo. You can click things like Add Row and Remove to change the grid. To view the code right click **test_edit_dijit.html** and choose **Open with > HTML Editor**

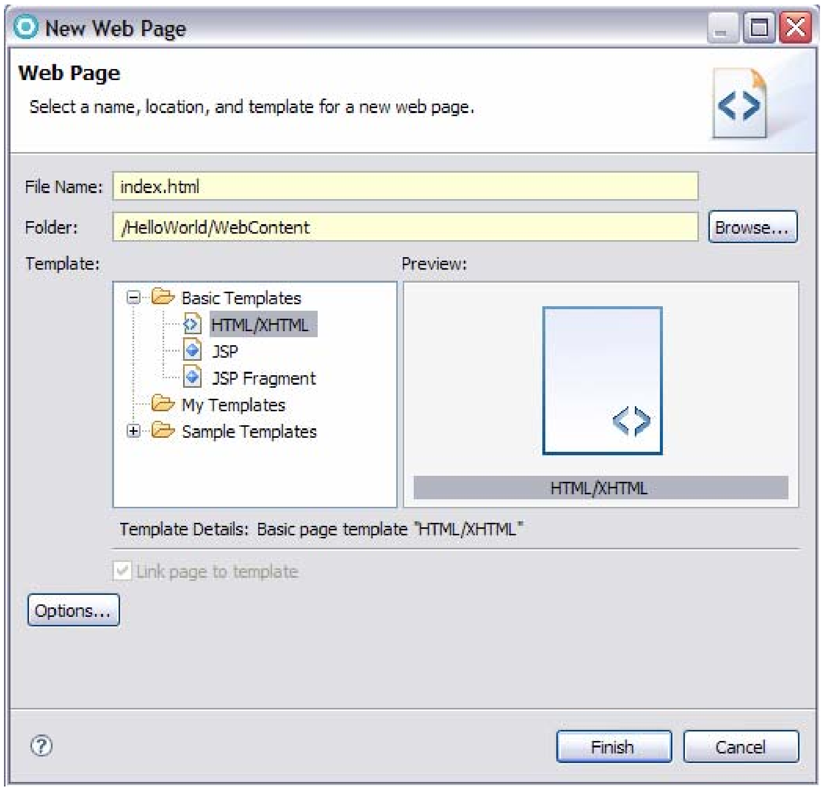
3) Select **dojo > dijit > tests > form** tests then right click **test_validate.html** and choose **Open with > Web Browser**. This html page shows the validation you can do on text boxes that is available to use with Dojo go to the Elevation and enter a number bigger than 20000 and see the instant validation on the text box. To view the code right click **test_validate.html** and choose **Open with > HTML Editor**

Part 2: Create client side of Hello World application

____ 1. Create a new Web page.

__ a. Right click the Web Content directory of your HelloWorld Application and choose **New > Web page**.

__ b. Enter **index.html** for the File name and choose **HTML/XHTML** for the Basic Template, Click **Finish**.


New Web Page

Web Page

Select a name, location, and template for a new web page.

File Name: index.html

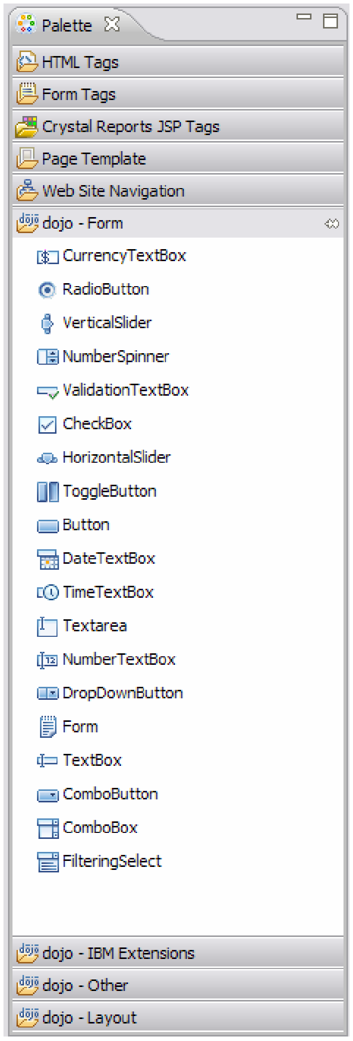
Folder: /HelloWorld/WebContent

Template: Basic Templates > HTML/XHTML

Template Details: Basic page template "HTML/XHTML"

____ 2. Your index.html page will be created and you should be in Web view. If you do not see a Palette on your right side of the Rational Application Developer then click the Web icon on the top right. The palette has different HTML tags and also dojo drag-and-drop tags.

____ 3. Click **dojo-Form** on the Palette



____ 4. Drag and drop the widget TextBox onto your index.html file after the <body> tag. You will notice that the correct dojo.require get added and your text box.

```
<input type="text" name="input1" value="value1"
dojotype="dijit.form.TextBox" trim="true" propercase="true">
```

__ a. Add an **id="Name"** tag within your text box widget so that you can look up the widget programmatically. Your widget will now look like:

```
<input type="text" name="input1" value="value1" id="Name"
  dojotype="dijit.form.TextBox" trim="true" propercase="true">
```

____ 5. Next **drag** the Button widget and drop it after your textbox widget.

```
<div dojotype="dijit.form.Button">Button</div>
```

__ a. Change the name of the button from Button to **Hello** and then add an **id="HelloButton"** to your widget.

```
<div dojotype="dijit.form.Button" id="HelloButton">Hello</div>
```

____ 6. You need to define an initialization function that gets references to the widgets (the name in the textbox field and the button) then define a function that will run whenever the button is pressed. The buttonPressed function gets the name that was entered in the textbox and then sends it as a JSON object to the server. The xhrArgs is the definition of what to call, and the object that is being sent to the server is the postContent. A POST is used so that the server will read the message out of the body of the request. The dojo.toJson function takes a JavaScript object and encodes it as JSON. You then invoke the xhr and then set the functions that will handle what to do when the xhr returns. In your case there is an alert with the response from the server and there is also an error catch.

__ a. **Paste** the following init function within the <script> tag in your index.html. The init function is a set up and event connection function. It uses dojo.AddOnLoad(), which is a register point for code that should run after dojo has fully loaded and all widgets parsed and instantiated. You cannot locate widgets with the function dijit.byId() until they have been parsed; hence why the event linking is done in init. What the init function ultimately does is attach the function called buttonPressed to the onClick of your button in the page. This means that any time you press the button, the buttonPressed function is called and data is posted to the server.

The code is also available in HelloSnippet1.txt

```
var init = function ()
{
   var nameBox = dijit.byId("Name");
   var button = dijit.byId("HelloButton");

   var buttonPressed = function ()
   {
         var yourName = nameBox.getValue();
         var xhrArgs = {
               url: "HelloServlet",
               handleAs: "json-comment-optional",
               postData: dojo.toJson({name: yourName})
         };
         var deferred = dojo.rawXhrPost(xhrArgs);
         deferred.addCallback(function (data){
               //Data is a json object with a property called message
               alert("Server replied: " + data.message);
         });
         deferred.addErrback(function (error){
               alert(error);
         });
   }
   dojo.connect(button,"onClick",buttonPressed);
}

dojo.addOnLoad(init);
```

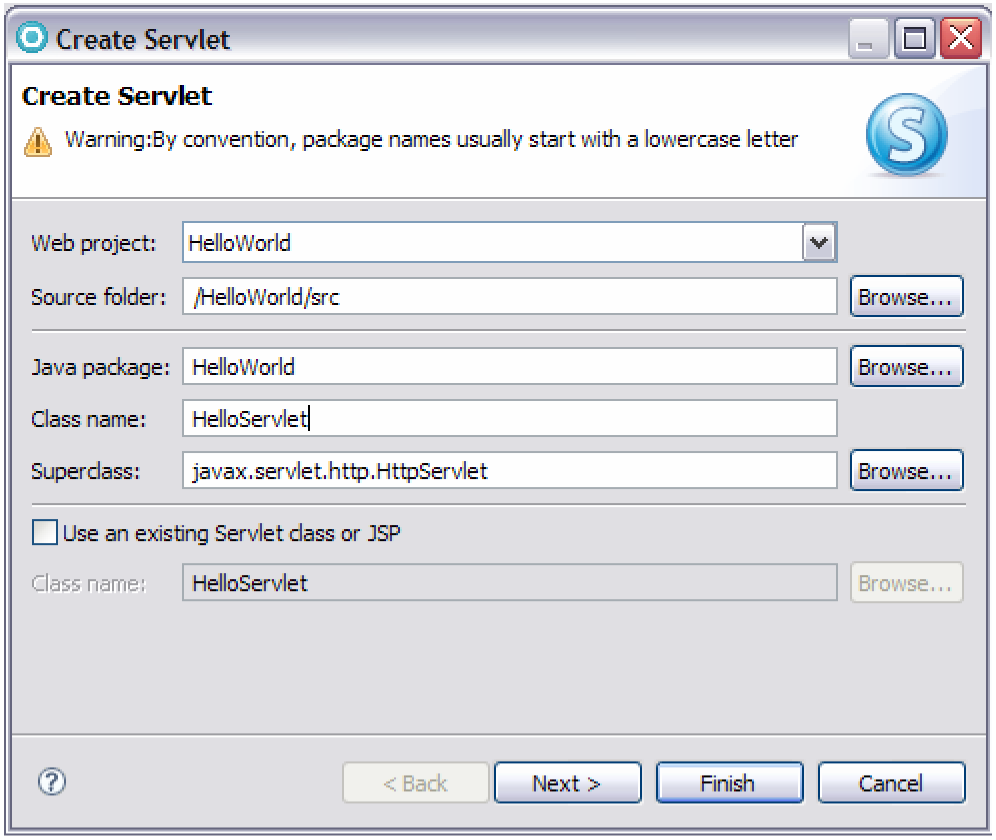
____ 7. **Save** your index.html file.

Part 3: Create server side of Hello World application

____ 1. Create a new servlet.

__ a. **Right click** the **Java Resources: src > Libraries** folder. Select **New > Servlet**.

__ b. Enter **HelloWorld** for the Java package and **HelloServlet** for Class name



Web project: HelloWorld
Source folder: /HelloWorld/src
Java package: HelloWorld
Class name: HelloServlet
Superclass: javax.servlet.http.HttpServlet
☐ Use an existing Servlet class or JSP
Class name: HelloServlet

__ c. Select **Next** then **Next and** make sure **doPost** is selected on the Create Servlet panel and you can clear doGet since you do not need it.

__ d. Select **Finish**

____ 2. In your HelloServlet.java add the json import

```
import com.ibm.json.java.*;
```

____ 3. In your HelloServlet.java you will add code into the doPost method. The code will get the body of the request and take JSON4J and parse the body and get your JSON object back. You then can easily find the name. You then check to make sure name is not null and write the response back. When you generate the reply you serialize the reply in comment-filtered JSON format.

__ a. **Replace** the doPost method in your HelloServlet.java with the following:

The code is also available in HelloSnippet2.txt

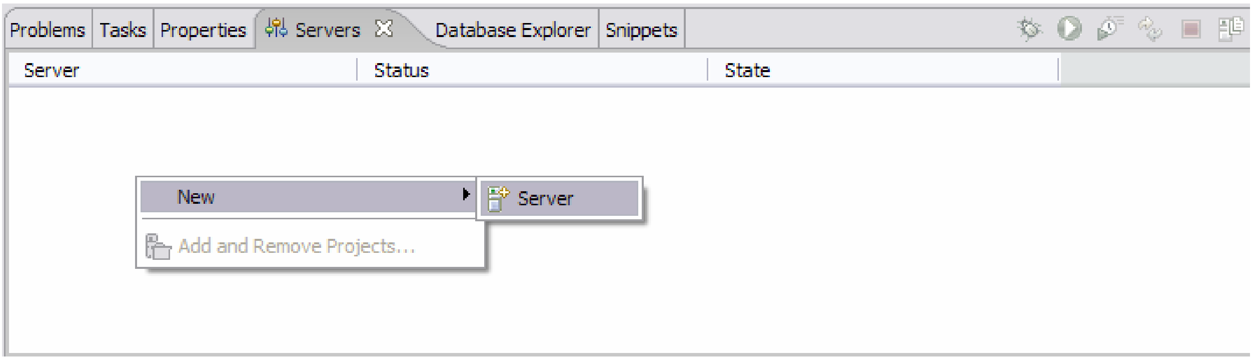
```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        try{
            JSONObject request = JSONObject.parse(req.getReader());
            if (request != null && request.get("name") != null) {
                JSONObject response = new JSONObject();
                response.put("message", "Hello, " + request.get("name"));
                resp.getWriter().write("/*" + response.serialize() + "*/");
            }else{
           throw new Exception("Content posted not in expected format.  The
body was null, or 'name' was not set");
            }
        } catch (Exception ex) {
            resp.sendError(HttpServletResponse.SC_UNSUPPORTED_MEDIA_TYPE,
ex.toString());
        }
    }
```

Part 4: Define your WebSphere Application Server V6.1

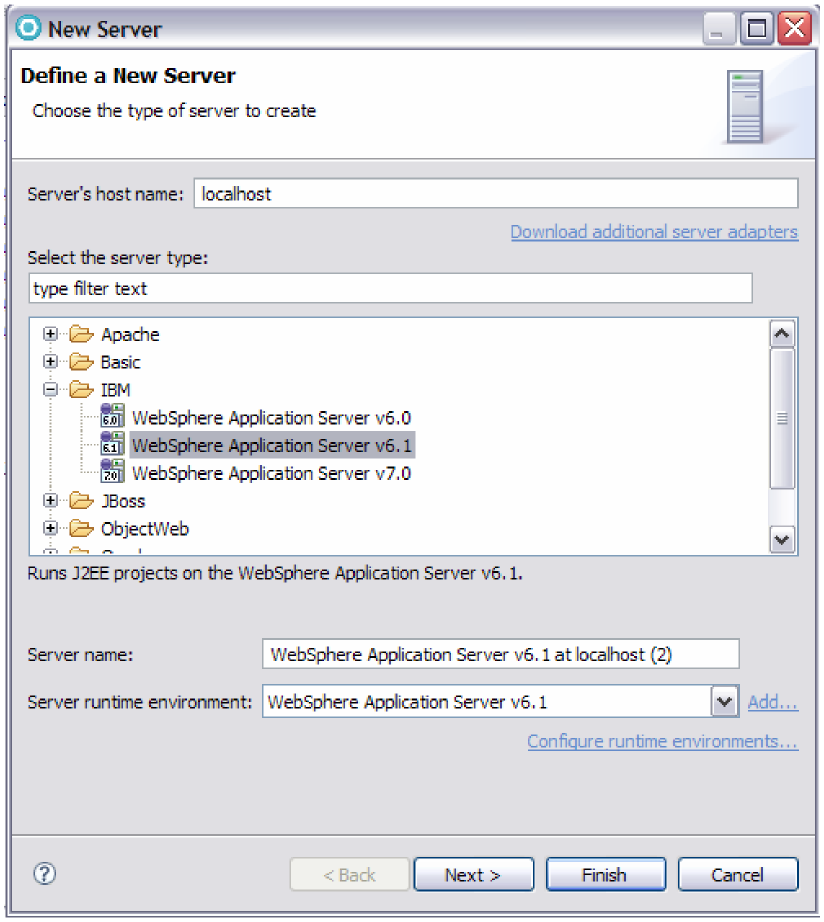
____ 1. **If a Server is not defined for your project yet**, add the WebSphere Application Server Version 6.1 If a WebSphere Application Server Version 6.1 is already defined skip to step 2.

__ a. In the lower panel, click the **Servers** tab.

__ b. Right-click in the lower panel and select **New > Server** from the contextual menu as shown below.



__ c. Select **IBM** > **WebSphere Application Server V6.1**


New Server
Download additional server adapters
Apache
Basic
IBM
WebSphere Application Server v6.0
WebSphere Application Server v6.1
WebSphere Application Server v7.0
JBoss
ObjectWeb
Add...
Configure runtime environments...
< Back
Next >
Finish
Cancel

Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

type filter text

Runs J2EE projects on the WebSphere Application Server v6.1.

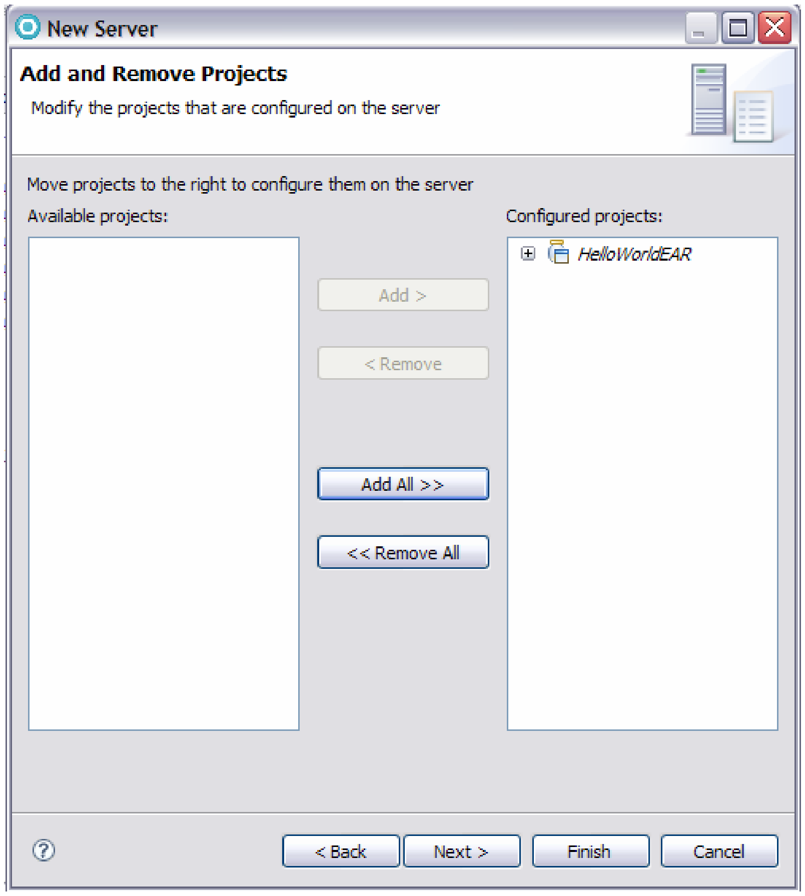
Server name: WebSphere Application Server v6.1 at localhost (2)

Server runtime environment: WebSphere Application Server v6.1

__ d. Click **Next**

__ e. Take the default settings and click **Next**

__ f. In the Add and Remove Projects select **HelloWorldEAR** and click **Add**



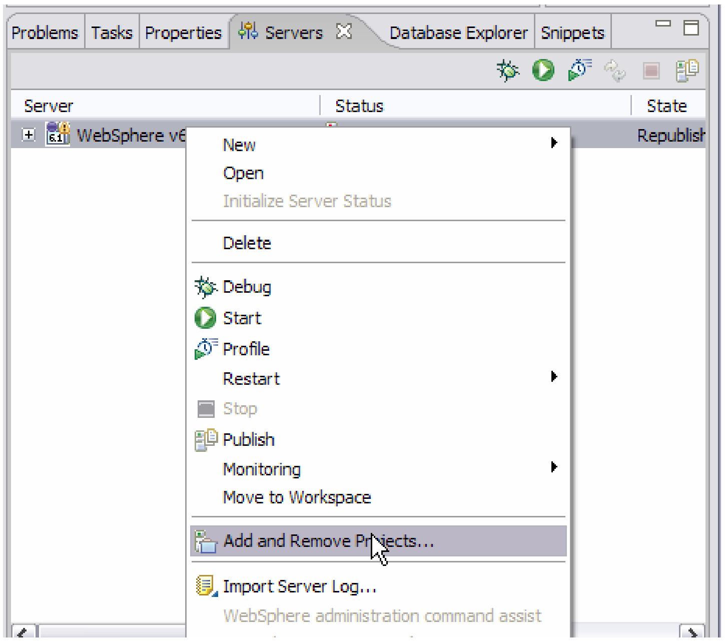
__ g. Click **Next**

__ h. Click **Finish**

____ 2. If you already have WebSphere Application Server V6.1 defined under Servers then just add the HelloWorld Project to the Server.

__ a. In the lower panel, click the **Servers** tab.

__ b. Right-click WebSphere Application Server V6.1 in the lower panel and select **Add and Remote Projects** from the contextual menu as shown below.



__ c. Select **HelloWorldEAR** and click **Add**

__ d. Click **Finish**

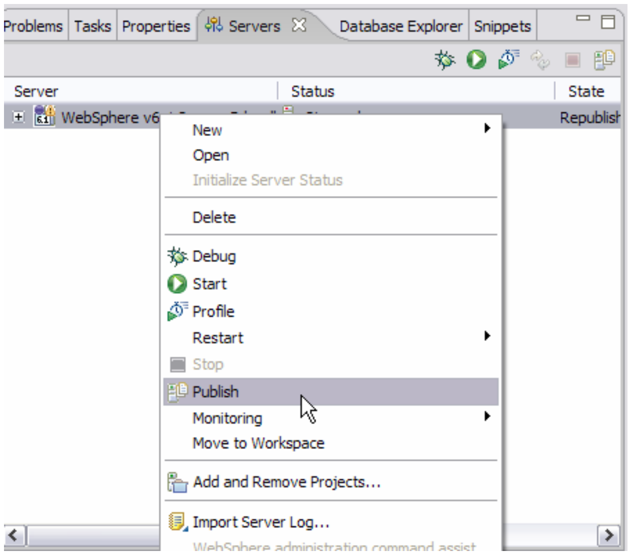
Note: this should be the only project associated with the server.

Part 5: Publish the Hello World application

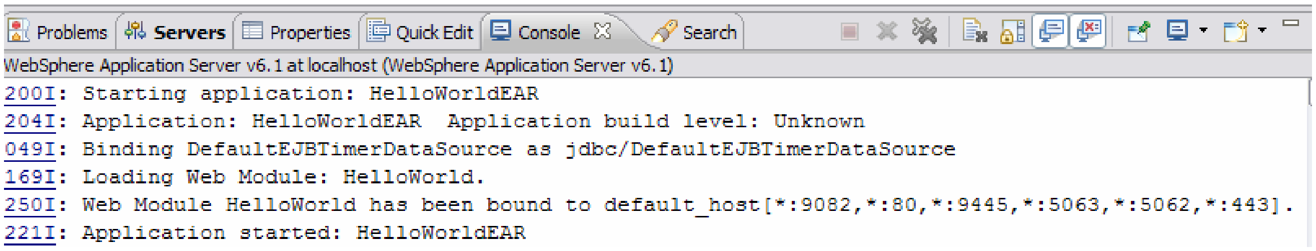
____ 1. Start the server and publish the HelloWorld application

__ a. In the lower panel, click the **Servers** tab.

__ b. Right-click WebSphere Application Server V6.1 in the lower panel and select **Publish** from the contextual menu as shown below. The server will start and the HelloWorld application will publish to the server too.



____ 2. In the console window you will notice that your HelloWorld Application will now be started. You can also view the console window to know the default port number.



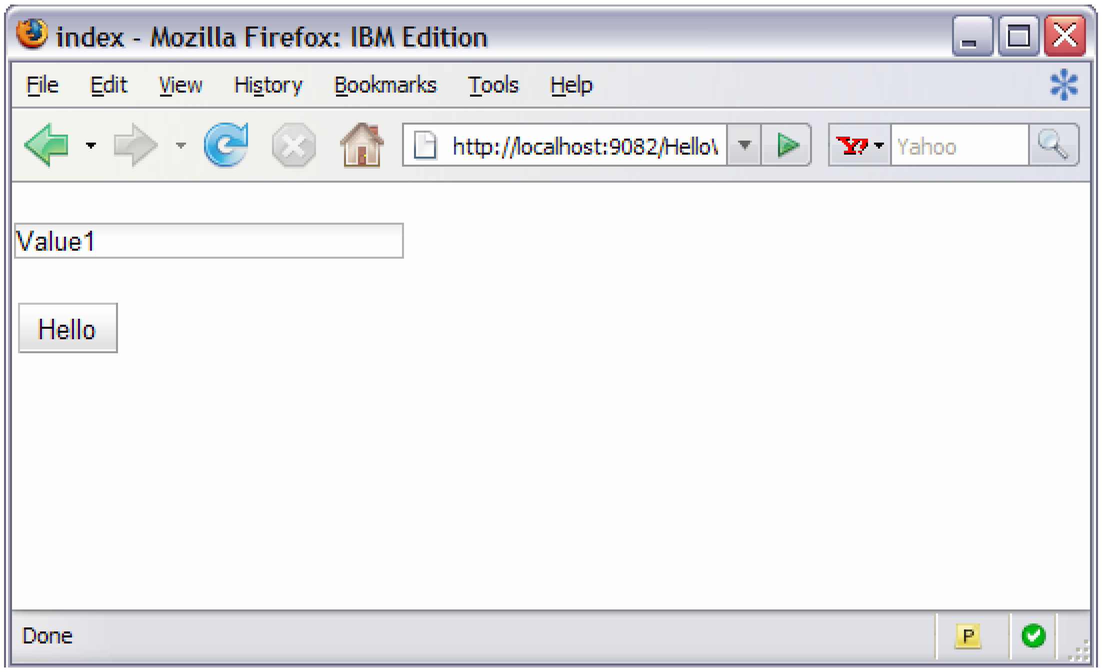
WebSphere Application Server v6.1 at localhost (WebSphere Application Server v6.1)

```
200I: Starting application: HelloWorldEAR
204I: Application: HelloWorldEAR  Application build level: Unknown
049I: Binding DefaultEJBTimerDataSource as jdbc/DefaultEJBTimerDataSource
169I: Loading Web Module: HelloWorld.
250I: Web Module HelloWorld has been bound to default_host[*:9082,*:80,*:9445,*:5063,*:5062,*:443].
221I: Application started: HelloWorldEAR
```

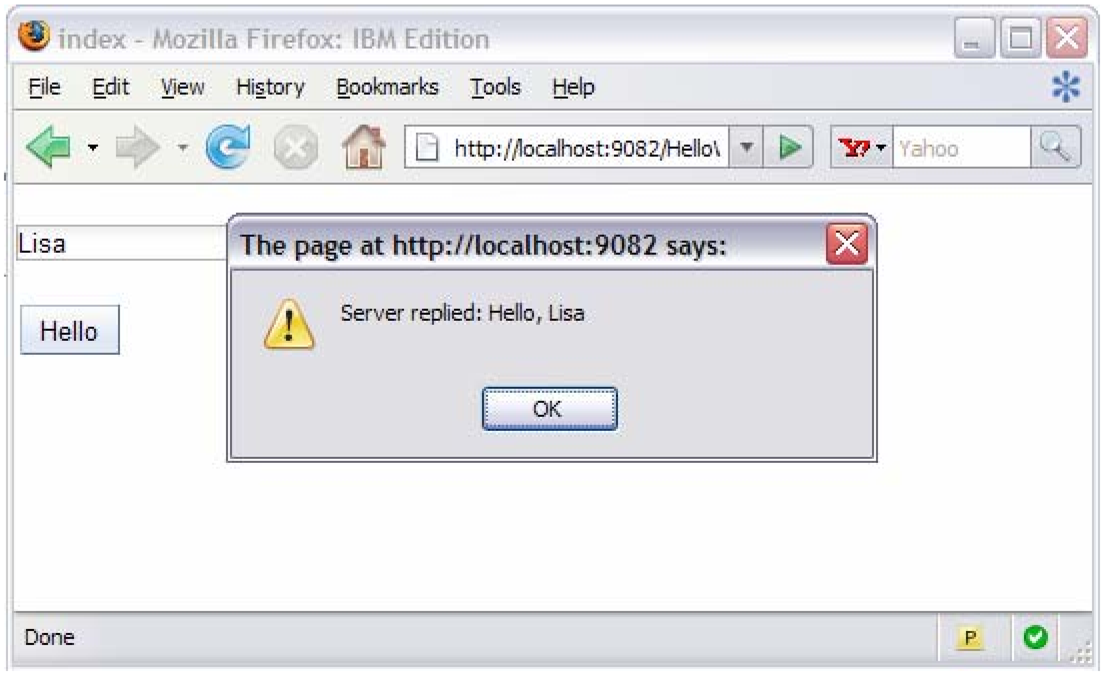
Part 6: Running the HelloWorld application

____ 1. Now you will view your HelloWorld application

__ a. View the application by opening a browser to http://localhost:9082/HelloWorld/ (note your port number might be different view the console to know your default port number)


index - Mozilla Firefox: IBM Edition
File
Edit
View
History
Bookmarks
Tools
Help
http://localhost:9082/Hello\
Yahoo
Value1
Hello
Done

____ 2. Enter a name into the textbox and select the Hello button.


index - Mozilla Firefox: IBM Edition
File
Edit
View
History
Bookmarks
Tools
Help
http://localhost:9082/Hello\
Yahoo
Lisa
Hello
OK
Done

The page at http://localhost:9082 says:

Server replied: Hello, Lisa

What you did in this exercise

In this lab you created a sample application that used dojo and the JSON4J library. The application was a simple HelloWorld application that had a textbox and button and replied back by saying Hello when the button was pressed.