

# Improved application development: Part 4, Building a Web client

Skill Level: Intermediate

[Nathaniel T. Schutta](#)  
Software developer  
Studio B

12 Jul 2005

In this tutorial, you return to the trusty Auction application and add a Web client. Taking advantage of powerful features in IBM Rational Application Developer, you develop a session bean facade without writing a single line of code. Once your session beans are in place, you'll use Rational Application Developer's intuitive Web designer to create pages that access your session beans, taking advantage of JavaServer Faces technology.

## Section 1. Before you start

### About this tutorial

This tutorial is the fourth in a five-part tutorial series. We strongly recommend that you take all the tutorials in this series sequentially, because each tutorial builds on something that was done in the previous tutorial.

In this tutorial, you'll return to the Auction application that you developed in Part 2. You'll add functionality to what you developed previously and connect to your entity beans via a Web-based front end. You'll take advantage of leading-edge technologies like JavaServer Faces (JSF) and Extensible Hypertext Markup Language (XHTML) to create a dynamic Web project -- and, thanks to IBM® Rational® Application Developer's powerful Web design features, you'll hardly have to touch the keyboard.

The tutorial includes the following:

- **Extending the Auction example:** You'll add additional functionality to the Auction application by taking advantage of Rational Application Developer's built-in code generation features. You'll also create a session bean from scratch, demonstrating the power of annotations.
- **The Web project:** Once your session beans are in place, you'll create a dynamic Web project. You'll use Rational Application Developer's powerful Web design capabilities, and you'll utilize modeling to help visualize the application.
- **Calling all session beans:** Next, you'll use JSF technology to call your session beans -- a process that is greatly simplified in Rational Application Developer.

Before you begin the tutorial, you should be familiar with the Java™ language and core Java 2 Platform, Enterprise Edition (J2EE) concepts. Experience with JSF and Enterprise JavaBeans (EJBs), although helpful, isn't required. You should also have completed the second tutorial.

## Prerequisites

To complete the steps in this tutorial, you should first have completed the Auction application from Part 2 of this series. You also need Rational Application Developer Version 6.0.

If you haven't already done so, you can download a free [trial version](#). Downloads are available for Microsoft® Windows® 2000, Windows Server® 2003, and Windows XP along with x86-based Linux machines.

---

## Section 2. Extending the Auction example

The second tutorial focused on the entities that make up the Auction example. Before you turn your attention to developing a Web front end to your entities, you need to extend the Auction application. In this section, you'll make those entities available to a Web application; but to do that, you need to add a couple of session beans to the mix. Rather than directly access your entity beans in the Web application, you'll use Service Data Objects (SDOs) and a session bean facade. Luckily, Rational Application Developer makes this easy.

**Why use a session bean?** If you're new to EJBs, you may ask why you're bothering with a session bean when you can access entity beans remotely. Think about the typical interaction a client has with data: a given form or page typically represents an entity bean (or two) and has inputs for the entity bean's fields. Imagine if you made a call to the server every time you entered something on the form -- performance would suffer as you made many fine-grained calls across the network. Session facades are used to improve performance by letting you make more coarse-grained calls to your entity beans. The session facade takes an abstraction of the data, often referred to as a *transfer object*. SDOs work together with JSF components to create Web applications that interact with data in a manner that is optimized for the Web.

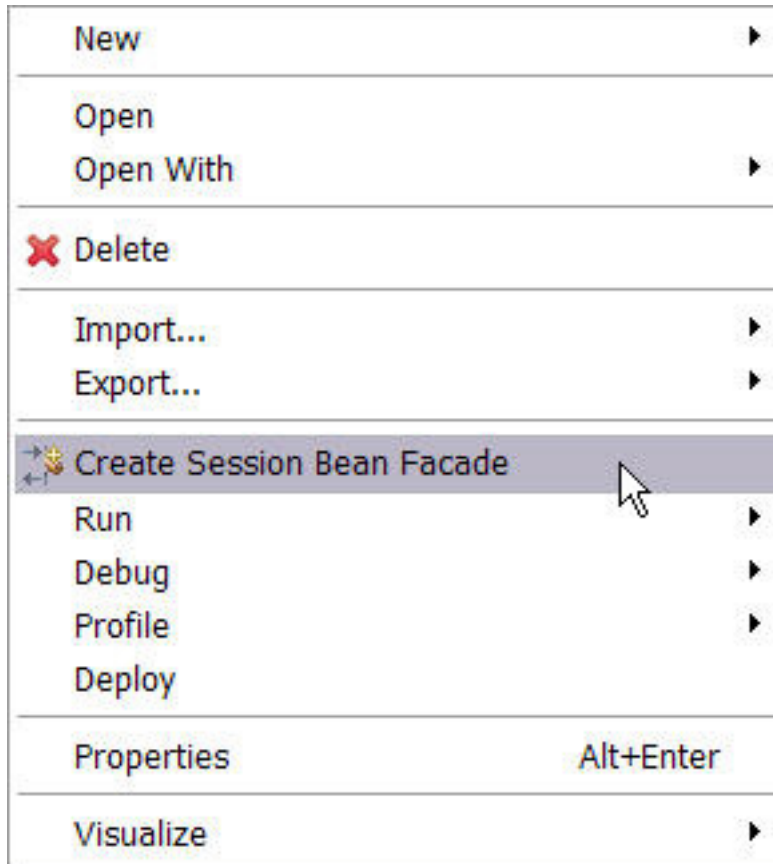
## Add the Item bean session facade

The first page that you'll create for your Web application is a list of items. To create the items list, you need a facade to your Item bean; later in this tutorial, you'll see how to leverage the Web design features of Rational Application Developer to connect to your facade. For now though, you'll focus on creating the necessary back-end code for your Web pages.

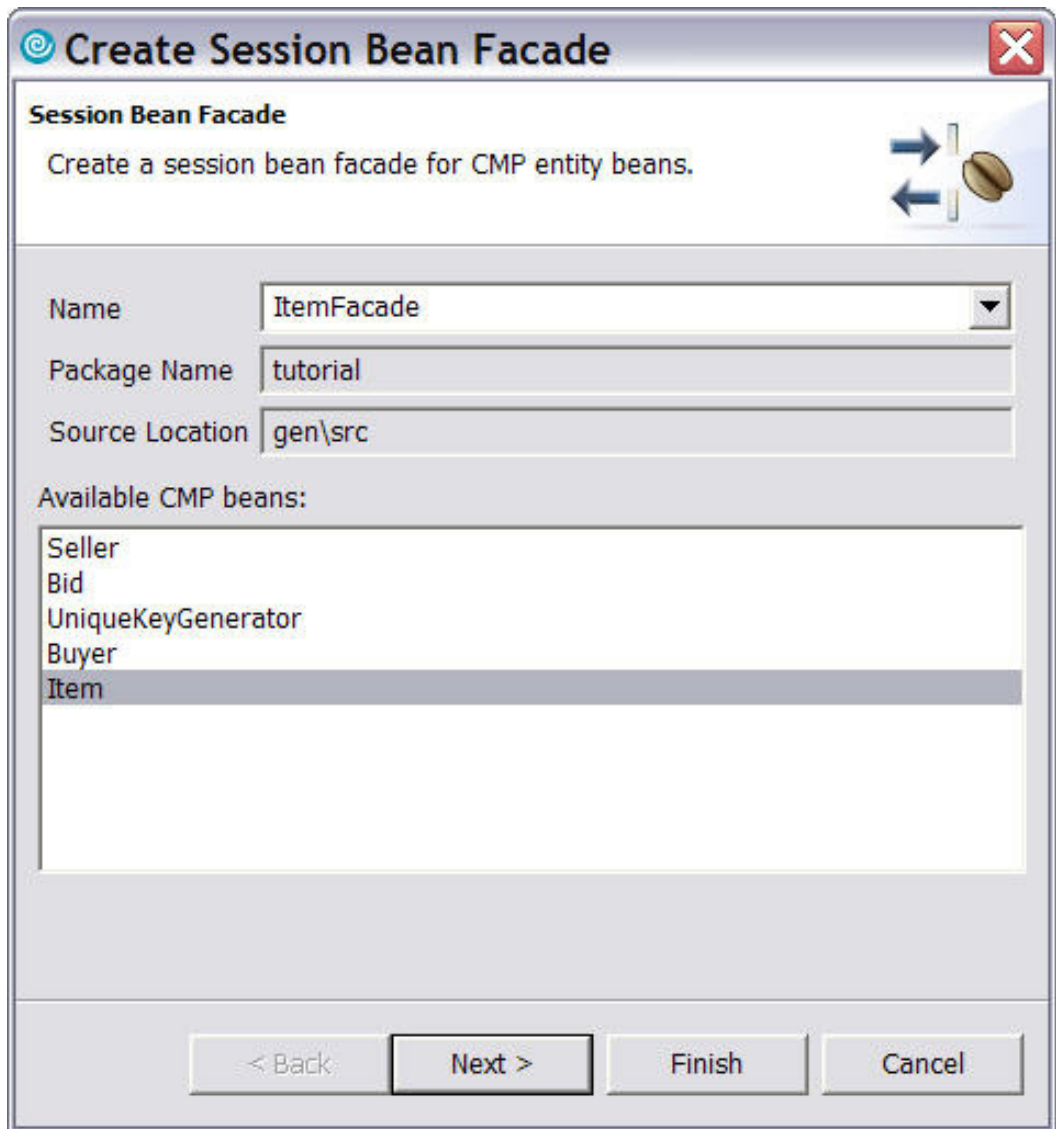
To create a session bean facade, you'll once again take advantage of the power of annotations and leverage the code-generation capabilities of Rational Application Developer. To create a session bean facade for your Item bean, perform the following steps from the Project Explorer view of the J2EE perspective:

1. Under EJB Projects, expand RAD tutorial.
2. Expand Deployment Descriptor: RAD tutorial.
3. Expand Entity Beans.
4. Right-click **Item**, and select **Create Session Bean Facade** (see Figure 1). Doing so launches the Create Session Bean Facade wizard (see Figure 2).

### **Figure 1. Create Session Bean Facade**

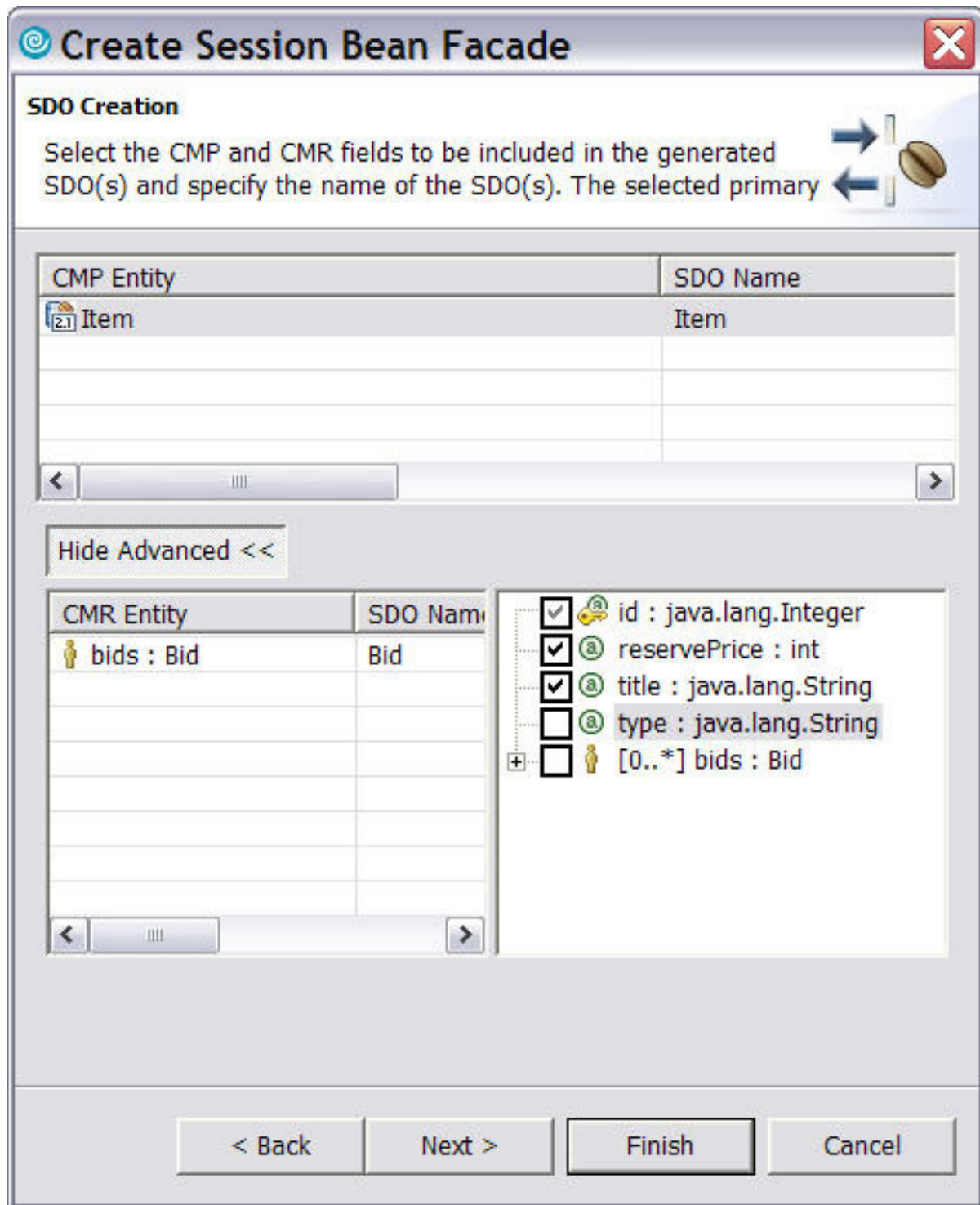


**Figure 2. Create Session Bean Facade wizard**



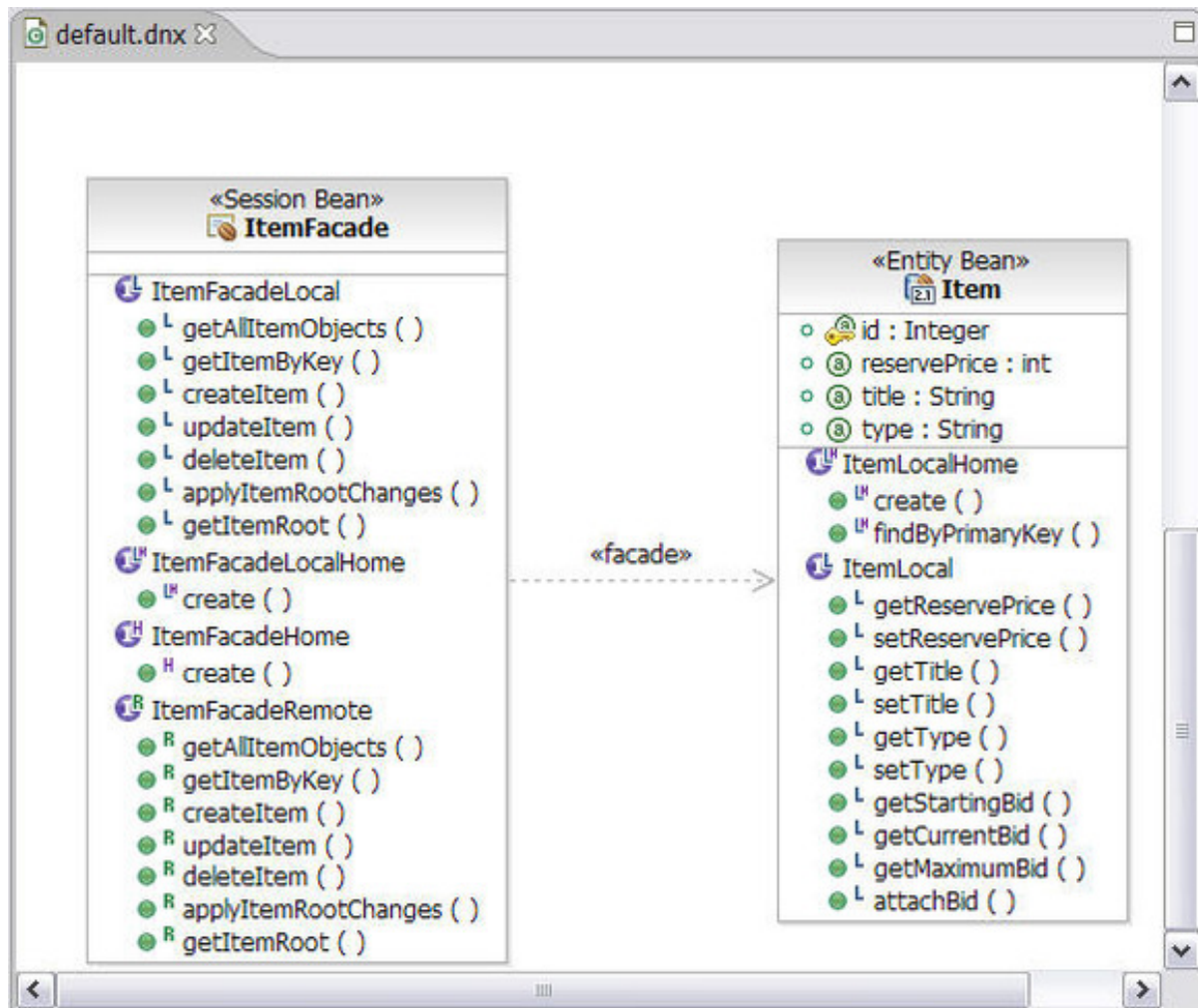
5. Make sure **Item** is selected, and click **Next**. The SDO Creation editor opens (see Figure 3).
6. Click **Show Advanced**.
7. Deselect **type**.
8. Click **Finish**.

**Figure 3. SDO Creation**



A class diagram opens, showing your new bean and its relationship to the Item entity bean (see Figure 4).

**Figure 4. ItemFacade class diagram**



## Creating the Auction service

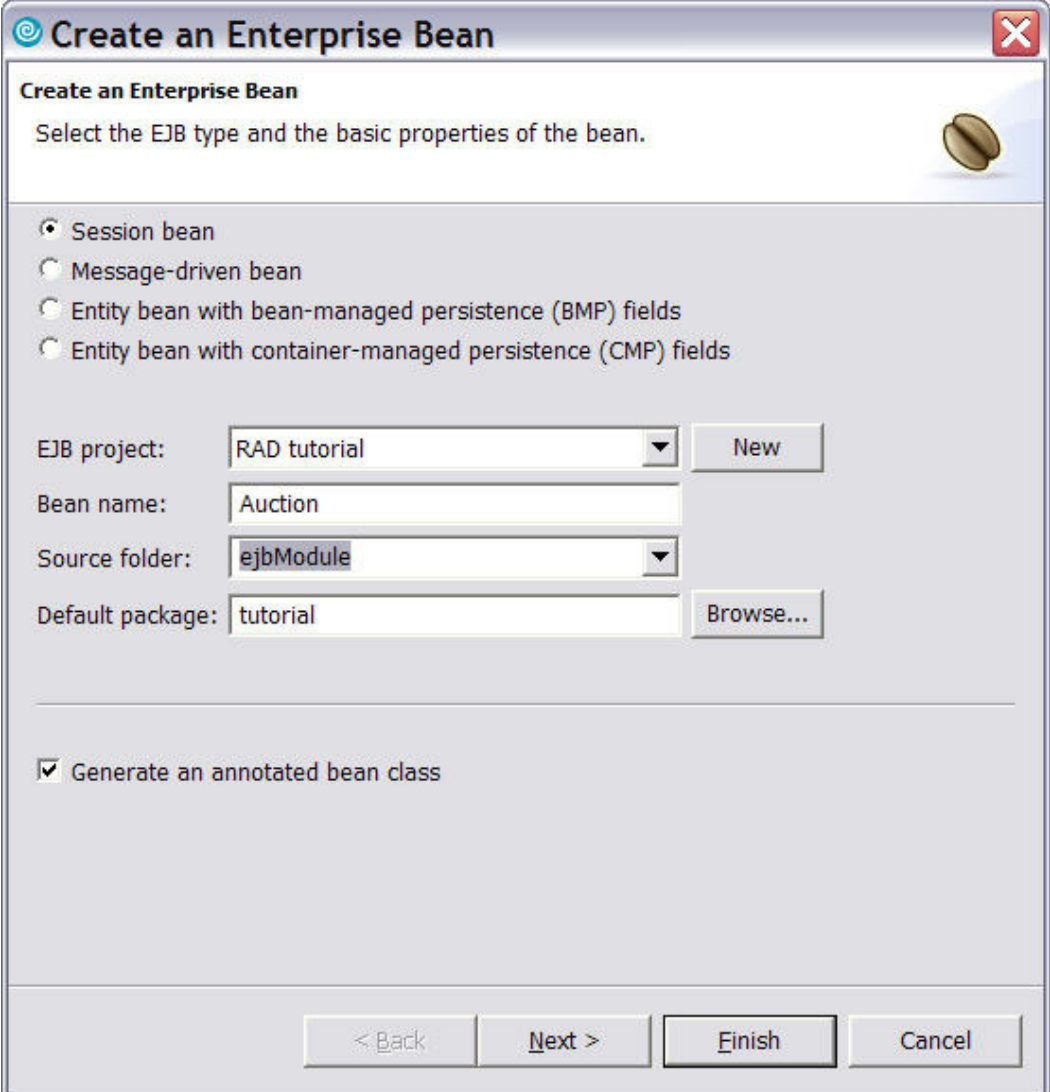
Your session facade allows you to perform useful actions like creating Items, finding Items, and updating Items, but your application wouldn't be very interesting if that's all it did. You want to be able to place a bid on a specific Item, which conceptually means you need to create a Bid and attach that Bid to the proper Item class. You also want to retrieve the highest bid and the current bid for a given Item. By now, these steps should look familiar.

To create the AuctionBean, follow these steps from the Project Explorer view of the J2EE perspective:

1. Under EJB Projects, expand RAD tutorial.
2. Expand Deployment Descriptor: RAD tutorial.

3. Right-click **Session Beans** and select **New > Session Bean** to launch the now-familiar Create an Enterprise Bean wizard.
4. Enter `Auction` for the Bean name.
5. Verify that the Source folder is `ejbModule`.
6. Change the Default package to `tutorial`.
7. Verify that **Generate an annotated bean class** is selected (see Figure 5).
8. Click **Next**.

**Figure 5. Create Session Bean window**



**Create an Enterprise Bean**

Create an Enterprise Bean

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

Session bean

Message-driven bean

Entity bean with bean-managed persistence (BMP) fields

Entity bean with container-managed persistence (CMP) fields

EJB project: RAD tutorial

Bean name: Auction

Source folder: ejbModule

Default package: tutorial

Generate an annotated bean class

< Back   Next >   Finish   Cancel

9. Make sure that **Remote client view** and **Local client view** are both selected.
10. Click **Finish**.

Now you have an AuctionBean, but frankly, it doesn't do much. The AuctionBean has a simple job: it takes a bid amount and applies it to an item. To do this, you need to retrieve the specific Item, create a Bid that supplies the information from the user, and then attach this newly created Bid to the proper Item:

1. In the Project Explorer view, under EJB Projects, expand RAD tutorial.
2. Expand Deployment Descriptor: RAD tutorial.
3. Expand Session Beans.
4. Expand Auction.
5. Double-click **AuctionBean** to open the AuctionBean within the Java editor.
6. Scroll down to the bottom and enter the code in Listing 1 for the methods.
7. Save your work by pressing Ctrl + S.

### Listing 1. AuctionBean code

```
/**
 * @ejb.interface-method
 */
public void placeBid(int itemKey, int bidAmount)
{
    try {
        javax.naming.InitialContext ctx = new javax.naming.InitialContext();
        BidLocalHome bidHome = (BidLocalHome)
            ctx.lookup("local:ejb/ejb/tutorial/BidLocalHome");
        BidLocal bidBean = bidHome.create();
        bidBean.setAmount(bidAmount);
        bidBean.setTimestamp(
            new java.sql.Timestamp(System.currentTimeMillis()));
        ItemLocalHome itemHome = (ItemLocalHome)
            ctx.lookup("local:ejb/ejb/tutorial/ItemLocalHome");
        ItemLocal itemBean =
            itemHome.findByPrimaryKey(new Integer(itemKey));
        itemBean.attachBid(bidBean);
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException();
    }
}

/**
 * @ejb.interface-method
```

```
*/
public int getHighestBid(int key)
{
    int highestBid = 0;
    try {

        javax.naming.InitialContext ctx = new javax.naming.InitialContext();
        ItemLocalHome itemHome = (ItemLocalHome)
            ctx.lookup("local:ejb/ejb/tutorial/ItemLocalHome");
        ItemLocal itemBean =
            itemHome.findByPrimaryKey(new Integer(key));
        BidLocal bid = itemBean.getMaximumBid();
        highestBid = bid.getAmount();
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException();
    }
    return highestBid;
}

/**
 * @ejb.interface-method
 */
public int getCurrentBid(int key)
{
    int currentBid = 0;
    try {

        javax.naming.InitialContext ctx =
            new javax.naming.InitialContext();
        ItemLocalHome itemHome = (ItemLocalHome)
            ctx.lookup("local:ejb/ejb/tutorial/ItemLocalHome");
        ItemLocal itemBean =
            itemHome.findByPrimaryKey(new Integer(key));
        BidLocal bid = itemBean.getCurrentBid();
        currentBid = bid.getAmount();
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException();
    }
    return currentBid;
}
}
```

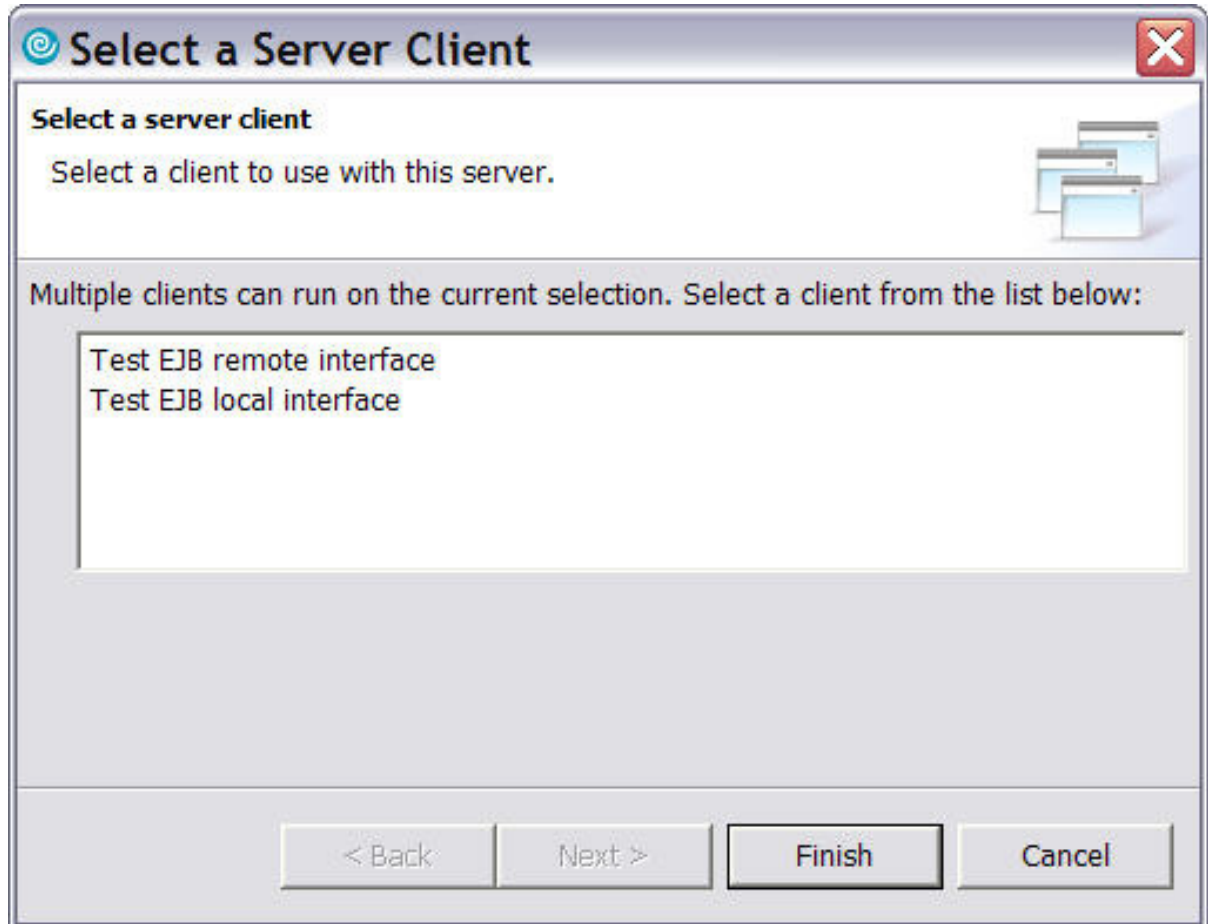
## Testing your beans

Before you go much further, you'd better test your new beans. To do this, use your old friend the Universal Test Client:

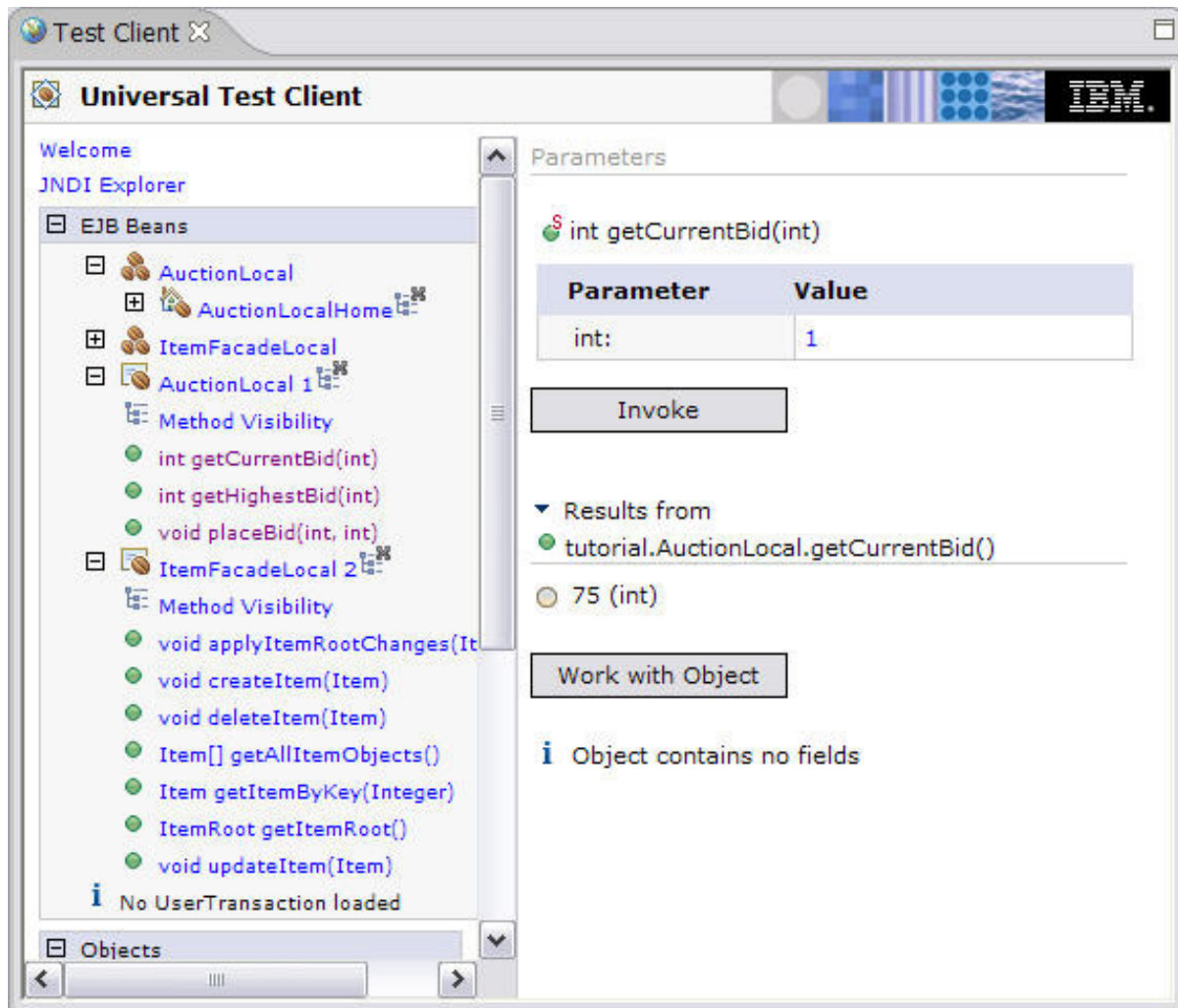
1. In the Project Explorer view, expand EJB Projects.
2. Expand Deployment Descriptor : RAD tutorial.
3. Expand Session Beans.
4. Right-click **Auction**, and then select **Run > Run on Server**.
5. Click **Finish** in the Server Selection window.

After the Universal Test Client opens, repeat steps 3 through 5 but select **ItemFacade**. Test your new beans: you should have an item or two from the previous tutorial. Make sure you test the three primary methods of the Auction service and the `getAllItemObjects()` method of the `ItemFacade`. Your session beans have both a remote and a local interface, so you'll be asked which interface to test; select **remote** when prompted (see Figure 6 and Figure 7).

**Figure 6. Select a Server Client window**



**Figure 7. Universal Test Client**



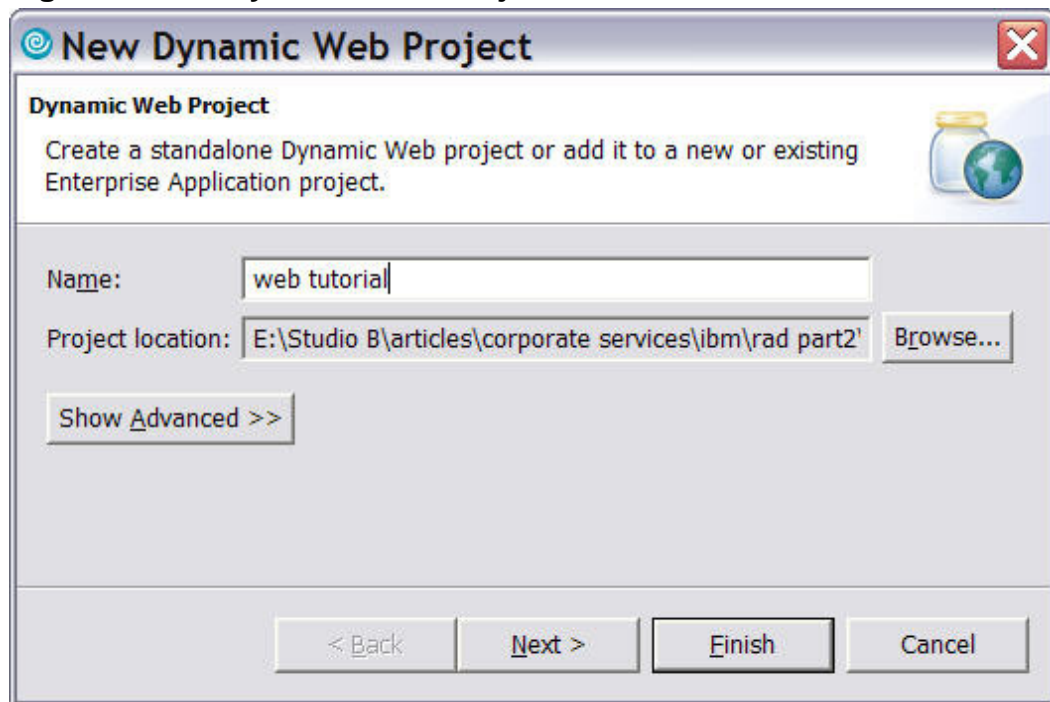
## Section 3. The Web project

Now that your session beans are in place, you can finally get to your Web application. To begin, create the Web project that will house your client code:

1. Select **File > New > Dynamic Web Project**. The menus are context sensitive; so, depending on which perspective is active when you select **File > New**, you might not see Dynamic Web Project. If Dynamic Web Project isn't visible, select **Other** and then **Dynamic Web Project**.
2. The New Dynamic Web Project wizard opens. Enter the name `web`

tutorial for your project in this example.

3. Leave the project location as is (see Figure 8).  
**Figure 8. New Dynamic Web Project wizard**

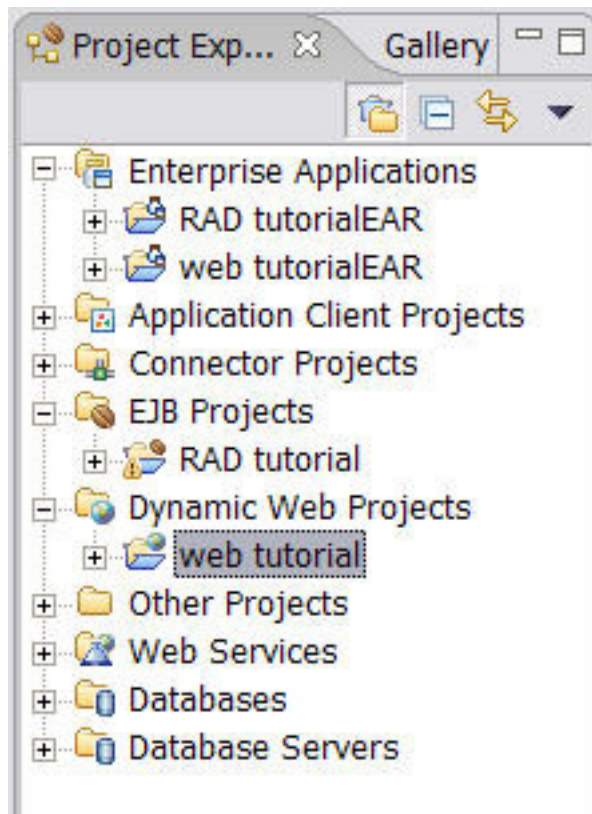


4. To develop to a different servlet version or target server, click **Show Advanced**.
5. Click **Finish**.
6. If you're prompted to switch perspectives, select **Yes**.

You now find yourself in the Web perspective. The Web perspective houses the views and editors you use to create and update Web resources such as JavaServer Pages™ (JSP), Hypertext Markup Language (HTML) files, and servlets. Wizards and content assist let you quickly and easily create complex static or dynamic Web content. Rational Application Developer includes support for leading-edge Web technology like JavaServer Faces and industry standard Web frameworks like Struts.

Notice that Rational Application Developer has now created a dynamic Web project and added web tutorialEAR to the Enterprise Applications folder (see Figure 9).

### Figure 9. Project Explorer



## Modeling your Web application

When you created your EJBs, you took advantage of the powerful modeling capabilities within the Rational Software Platform. This allowed you to visually create your entities and greatly simplified the development process. In Rational Application Developer, modeling isn't just for entity beans; you can also model your Web application. You can quickly and easily define the components and navigational flow of your application before you get hip-deep in code.

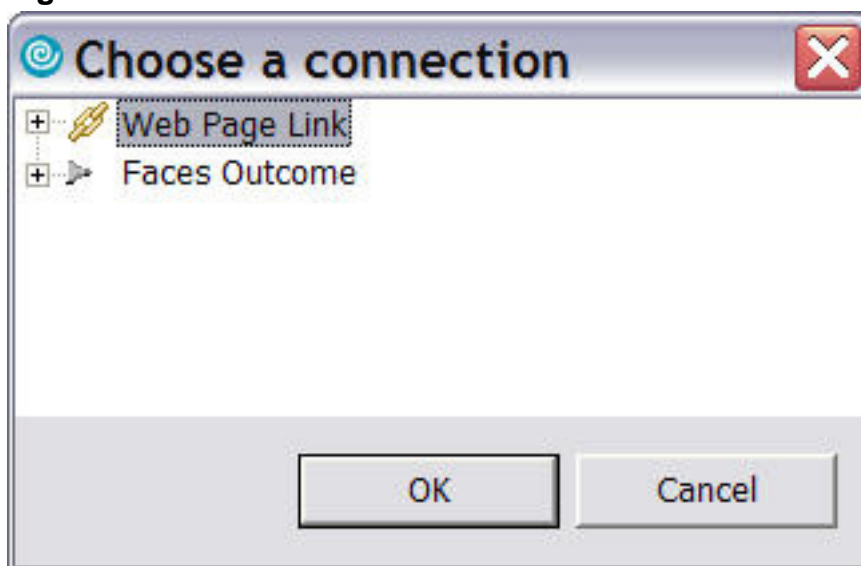
From the Project Explorer view of the Web perspective:

1. Expand Dynamic Web Projects.
2. Expand web tutorial.
3. Double-click **Web Diagram** to open the Web Diagram editor.
4. From the Palette view, select **Web Page** and click the **Web Diagram editor**.
5. Name the jsp icon `/itemList.jsp`, and press Enter (note that the

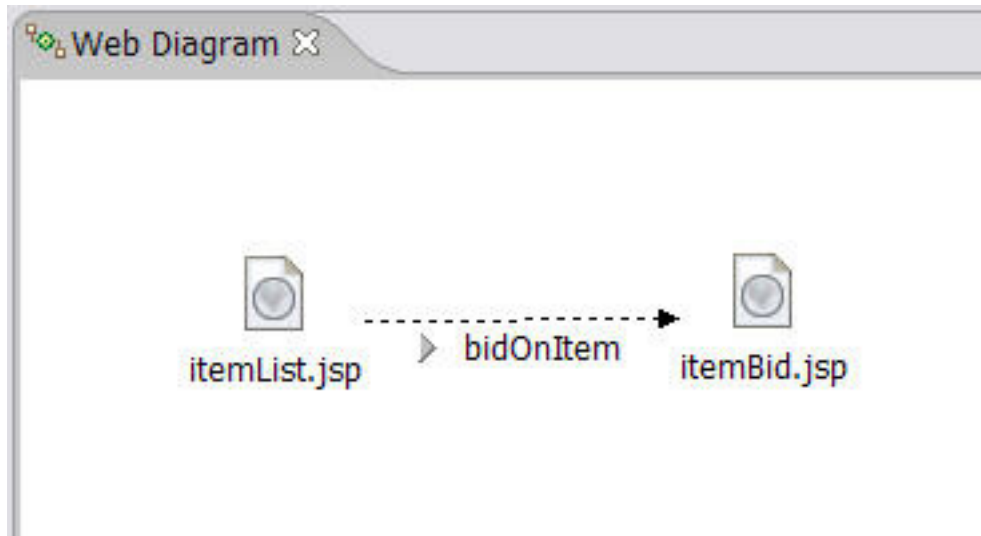
forward slash will disappear).

6. Repeat steps 4 and 5 to add `/itemBid.jsp`.
7. From the Palette view, select **Connection**.
8. Click **itemList.jsp**, and then click **itemBid.jsp**. Doing so opens the Choose a connection window (see Figure 10).

**Figure 10. Choose a connection window**



9. Expand Faces Outcome, click **<new>**, and then click **OK**.
  10. Rename `<new>` to `bidOnItem`.
  11. Save the diagram by pressing `Ctrl + S` (see Figure 11).
- Figure 11. Web diagram**



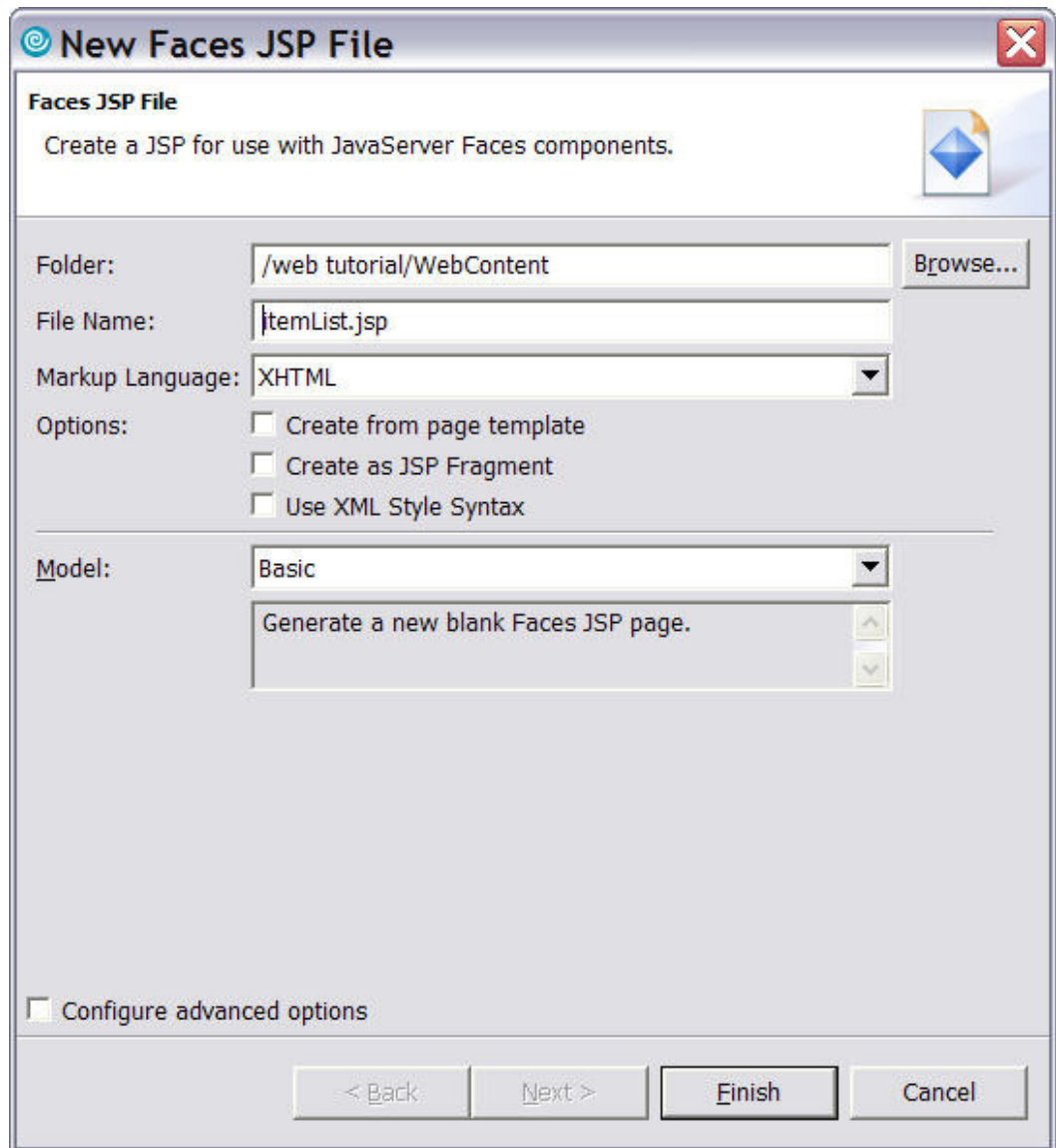
Obviously, this is a trivial example, but imagine if you had a more complex application. The ability to model your pages graphically, adding and removing connections with ease, is a powerful feature of Rational Application Developer.

## Generating code

Now that you've expressed your application visually, you can use your model to generate pages. Although doing so isn't quite as automatic as the process of creating your entity beans, you won't have to perform much heavy lifting. If you've spent any time working with Web pages, you understand how much coding is involved in even simple applications. Luckily for you, Rational Application Developer is well equipped to simplify the development cycle.

To create a JSP:

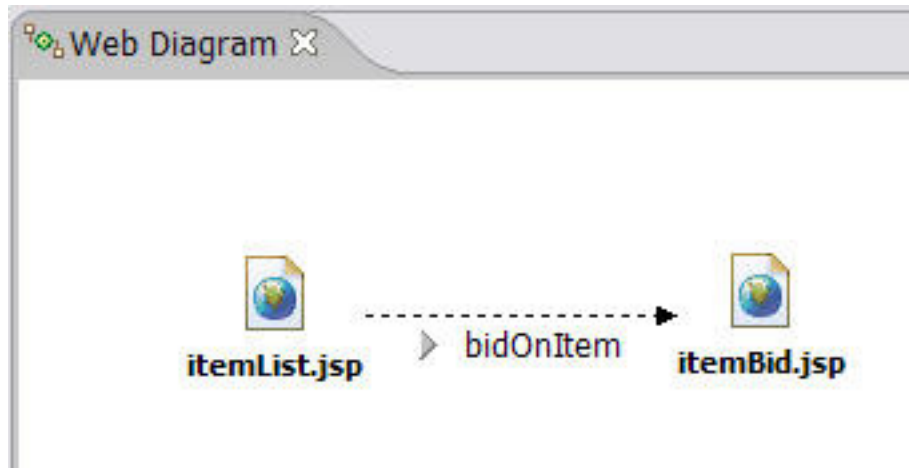
1. Double-click **itemList.jsp** in the Web Diagram editor. Doing so opens the New Faces JSP File window.
2. Change the Markup Language to **XHTML** (see Figure 12).  
**Figure 12. New Faces JSP File wizard**



3. Click **Finish**.
4. Return to the Web Diagram, and double-click **itemBid.jsp**. Don't forget to change the Markup Language to **XHTML**.

Notice that your Web Diagram looks a little different now: the icons are in color, and the names are in bold print. This change indicates that those pages are *realized* -- in other words, an actual file is associated with the icon (see Figure 13). This feature can be helpful when you're working with a large model.

**Figure 13. Web Diagram with realized icons**



**What is XHTML?** If you guessed from the acronym that XHTML is a combination of XML and HTML, you're correct. XHTML takes the properties of XML -- that documents be well formed and correctly marked up (elements must be properly nested and closed) -- and combines them with the presentation ability of HTML. Although it isn't universally used, XHTML has been the official HTML standard of the World Wide Web Consortium (W3C®) since 26 Jan 2000.

**Using JSF:** You're going to leverage the power of JavaServer Faces (JSF), a new technology that simplifies the creation of Web applications. JSF lets you assemble Web pages from reusable user interface components that can be easily connected to data sources. You can think of JSF as a cross between Struts (the popular open source Web framework) and Swing. JSF provides a rich set of graphical user interface (GUI) components to use in creating complex Web applications.

At this point, you need to edit the navigation rule between `itemList.jsp` and `itemBid.jsp`:

1. Return to the Web Diagram.
2. Double-click the **bidOnItem connection** label to open the Edit Navigation Rule window.
3. Leave the defaults, and click **OK** (see Figure 14). Notice that your connection label is now in bold.

**Figure 14. Edit Navigation Rule window**

**Edit Navigation Rule**

Select the pages to be used in this rule:

From page: /itemList.jsp

To page: /itemBid.jsp

When the action returns the outcome:

Any outcome

The outcome named: bidOnItem

This rule is used by:

Any action

This action only:

When following this rule:

Use request forwarding (parameters work automatically)

Use request redirection (parameters must be coded)

OK Cancel

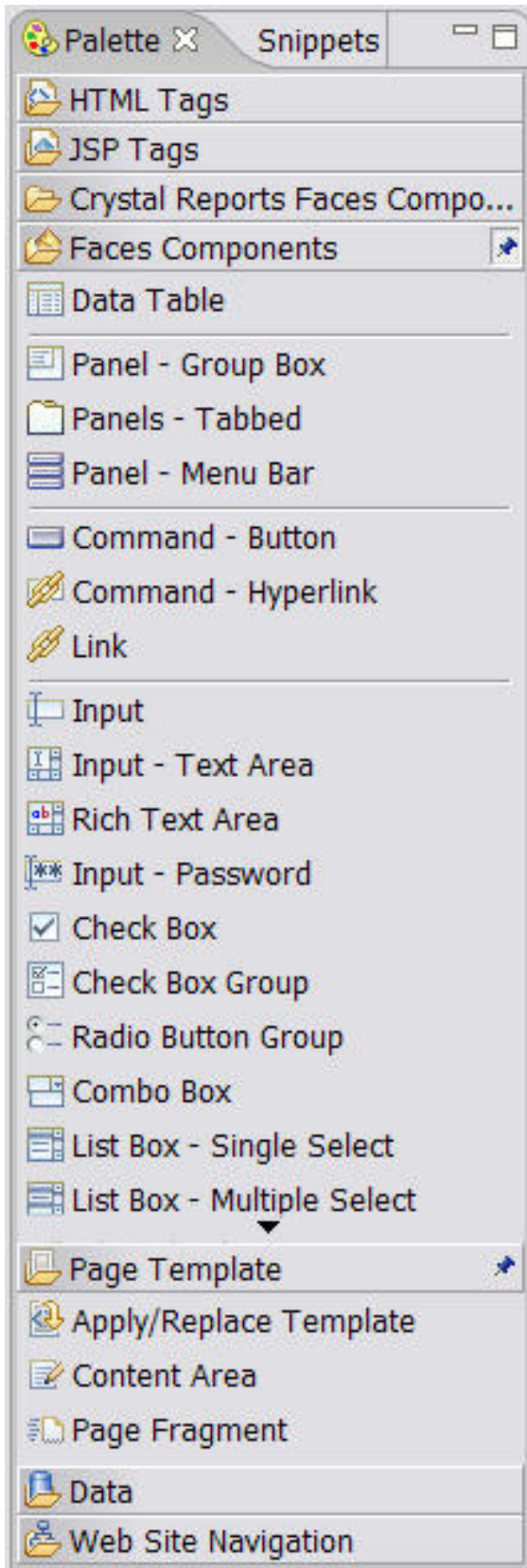
## The Web perspective

When you clicked Finish on the New Faces JSP File wizard, the files opened in the Page Designer. Notice that the editor has three tabs: Design, Source, and Preview. These tabs allow you to work with your pages in a WYSIWYG-style editor or directly in the page's source code, depending on your preferences. The Preview tab gives you an idea of what your work will look like in a browser.

Also notice that your Palette has come to life. It includes drawers for HTML Tags, JSP Tags, Faces Components, and other common Web features (see Figure 15). Within these drawers are components you use often, like Check Box, Input fields,

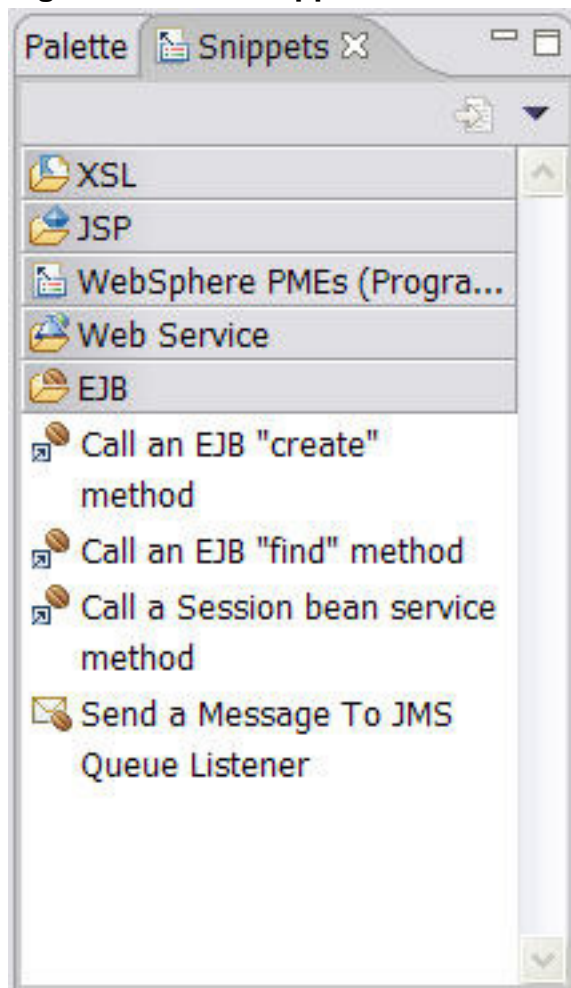
Image tags, Table tags, and standard JSP tags like Choose and When, all at your disposal. Rather than focus on the monotonous coding of your three-hundredth a `href`, you can spend your time concentrating on designing an intuitive interface and meeting your system requirements.

### **Figure 15. Web Palette**



Take note of the Snippets view, which contains a number of useful code fragments (see Figure 16). How often have you copied and pasted something from an existing Web page? Have you ever forgotten the `modify` part of `cut-paste-modify`? Now you can take advantage of these common code snippets to further improve your productivity. As you'll soon see, you even find help connecting to EJBs.

**Figure 16. Web Snippets**



## The Page Designer

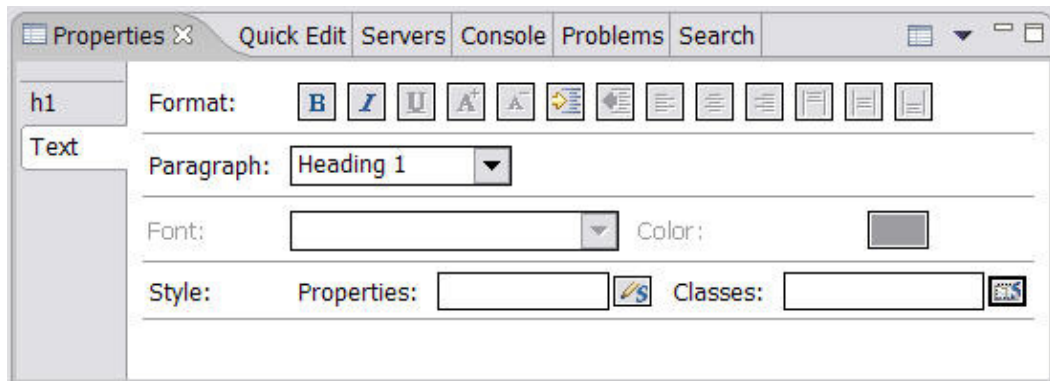
Take a moment to familiarize yourself with the WYSIWYG editor. Click the `itemList.jsp` tab in the editor window. Notice that the new page has a bit of text already; although it's a helpful reminder that you have work to do, it's not exactly what you want for the title of your list page. Start by deleting the default text and replacing it with something more intuitive, like `Items List Page`.

Of course, titles usually aren't just plain text, so let's change the properties of the text

to make it bigger. To modify the settings of your title, perform the following steps:

1. Highlight your title text.
2. Select the Properties view (see Figure 17).

**Figure 17. Properties view**



3. Verify that **Text** is selected.
4. Select **Heading 1** from the Paragraph drop-down menu.

Voila: Your title is now a heading. As you can see, this is only the beginning of what you can do with the Design tab. If you were using Cascading Style Sheets (CSS), you could use the Classes editor to apply them easily to your pages. You can use familiar icons to apply specific formatting to text. You can rapidly change the background color or change the page margin, all without touching the actual code for the page. Not only is this a great productivity boost for new Web developers, but it can also be useful for those of you who are accustomed to using text editors. Rather than scour the Web or a dog-eared copy of a trusted reference manual, you can make a change using the Design tab and then switch to the Source tab to look at the proper markup. Rational Application Developer's Web design features are a powerful addition to your development toolbox.

---

## Section 4. Calling all session beans

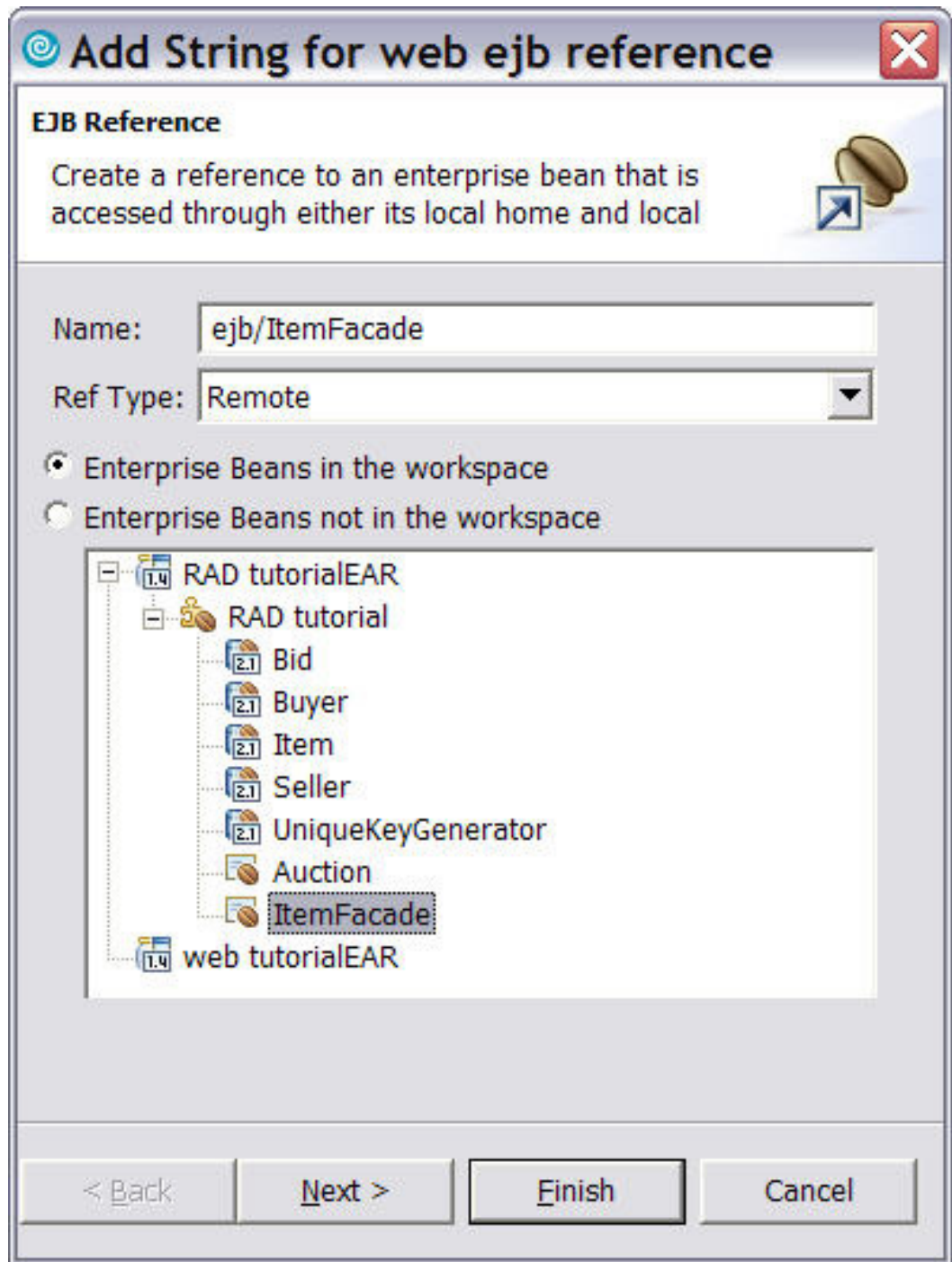
Static text is fine and good, but for this application to be of any interest, call the session beans you created earlier. As you've noticed from the code you've added to the beans, quite a bit of overhead is associated with entity beans. You have to get an initial context, look up the home interface, and then get an instance of the

component interface, all before you can do any interesting work. This code is repetitious when you're working with multiple beans, and it's easy to make a mistake (especially if you take the cut and paste shortcut -- and who doesn't?). Luckily, Rational Application Developer takes care of this rote work, once again freeing you to focus on more creative aspects of application development.

Let's start with your list of Items:

1. Expand the Data drawer in the Palette view.
2. Click **EJB Session Bean**.
3. Click below the title on **itemList.jsp** to open the Session Bean window.
4. Click **New EJB Reference** to open the EJB Reference window (see Figure 18).

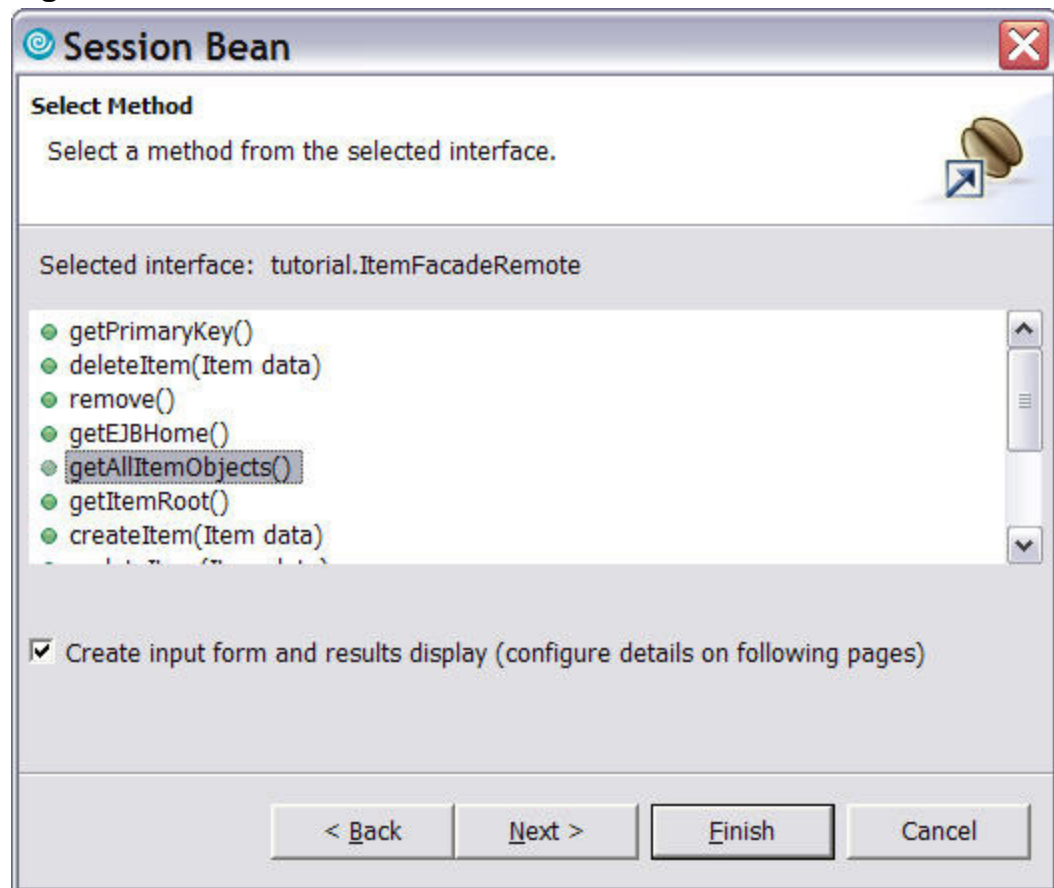
**Figure 18. EJB Reference window**



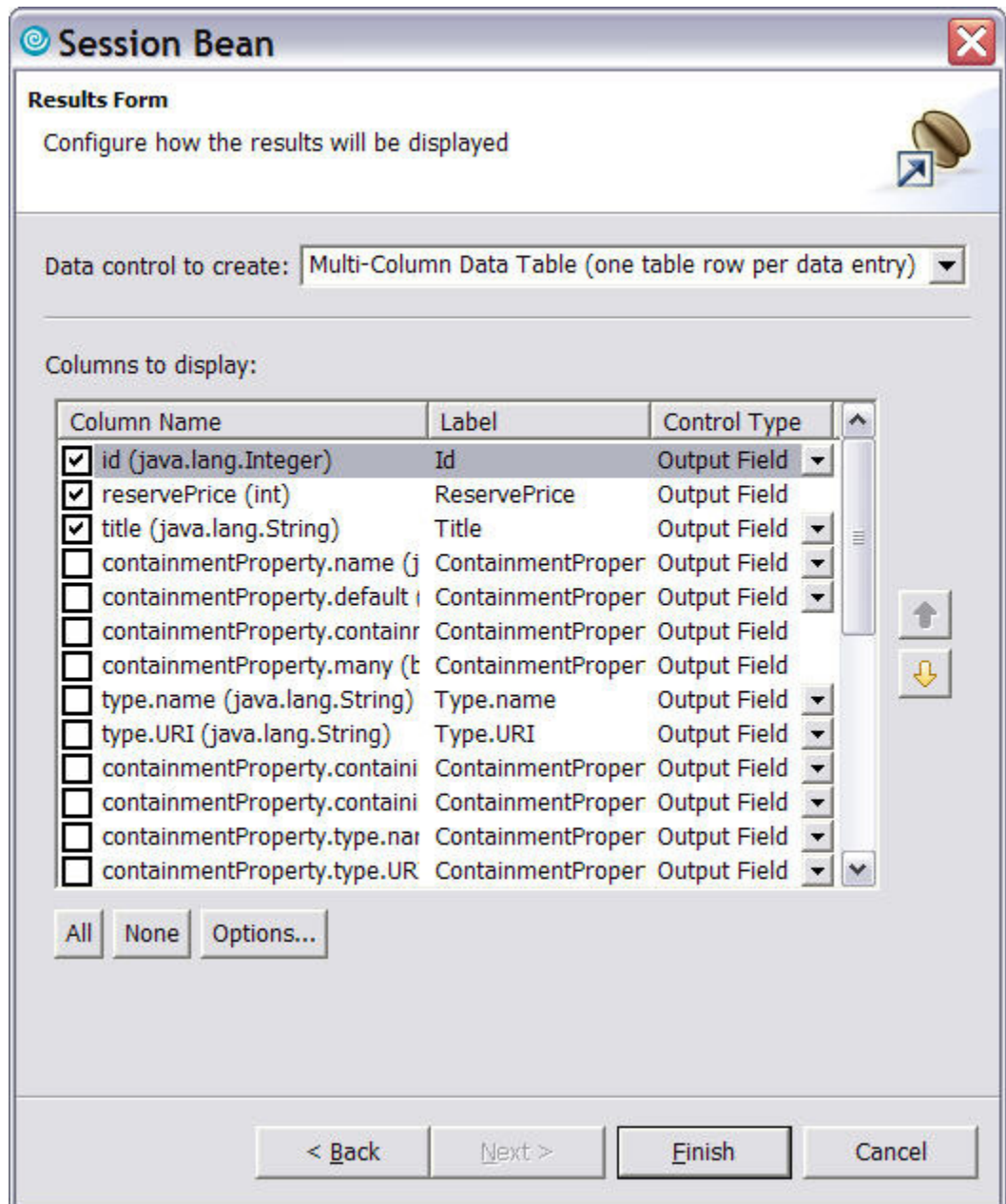
5. Ensure that **Enterprise Beans in the workspace** is selected.
6. Expand RAD tutorialEAR.
7. Expand RAD tutorial.

8. Select **ItemFacade**.
9. Click **Finish** to return to the Session Bean window.
10. Click **Next**.
11. Select **Use default context properties for doing a lookup on this reference**.
12. Click **Next**.
13. From the Select Method window (see Figure 19), select **getAllItemObjects()**. Be sure **Create input form and results display** is selected.

**Figure 19. Select Method window**



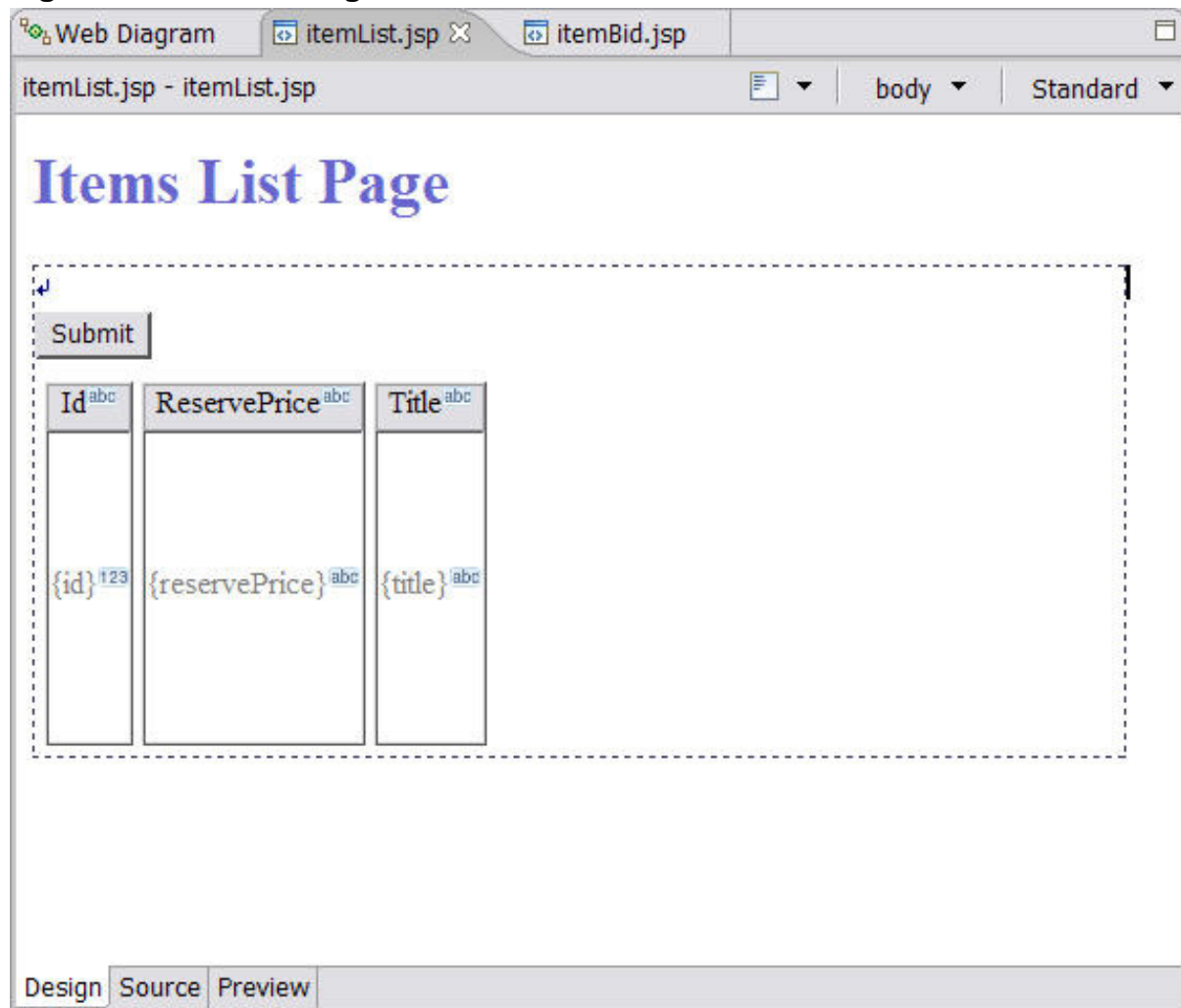
14. Click **Next**. The Results Form window opens (see Figure 20).
- Figure 20. Results Form window**



15. As you can see, you can pick and choose from a number of attributes. Click **None**.
16. Select **id**, **reservePrice**, and **title**.
17. Click **Finish**.
18. When the editor returns to focus, save your work by pressing Ctrl + S.

Whew -- you just did a lot. Your itemList.jsp should now look something like Figure 21.

**Figure 21. Item List Page with a table**



In case you don't think Rational Application Developer was very helpful, switch to the Source view and look at the code that was generated on your behalf.

## Modifying your list page

Rational Application Developer did a lot for you, but you have to make a few minor modifications. First, you should rename the button -- Submit isn't very meaningful. Perform the following steps from Design view:

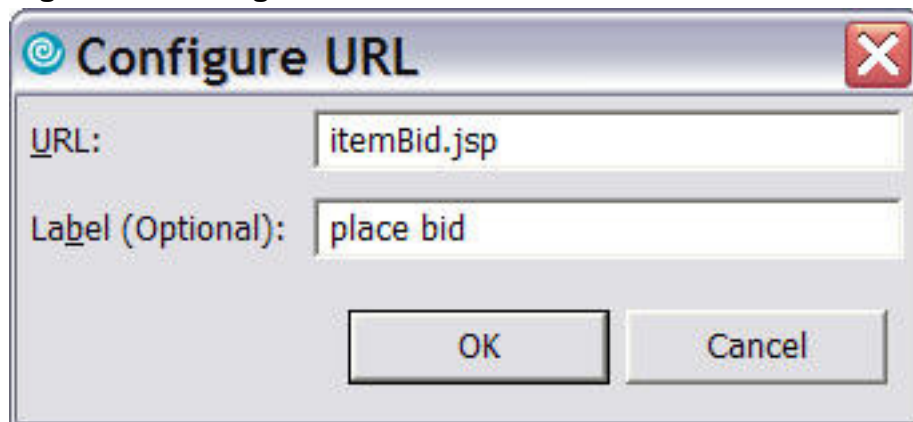
1. Select the **Submit** button.


2. In the Properties view, select the **Display options**.
3. Change the button label to `Refresh List`.
4. Press Enter.


Now you need a way to bid on an Item. Add a column to your table and add a link to the `itemBid.jsp` that includes the id for the Item you're bidding on. From the design view:

1. Click any of the column headers. In the Properties view, select **h:dataTable**.
2. On the far right of the Properties view is the Column data. Click **Add** to add the new column.
3. Type `Action` for the new column name, and press Enter.
4. In the Palette view, expand the Faces Components drawer by clicking it.
5. Select the **Link component**, and place it on the column you just created. The Configure URL window opens (see Figure 22).

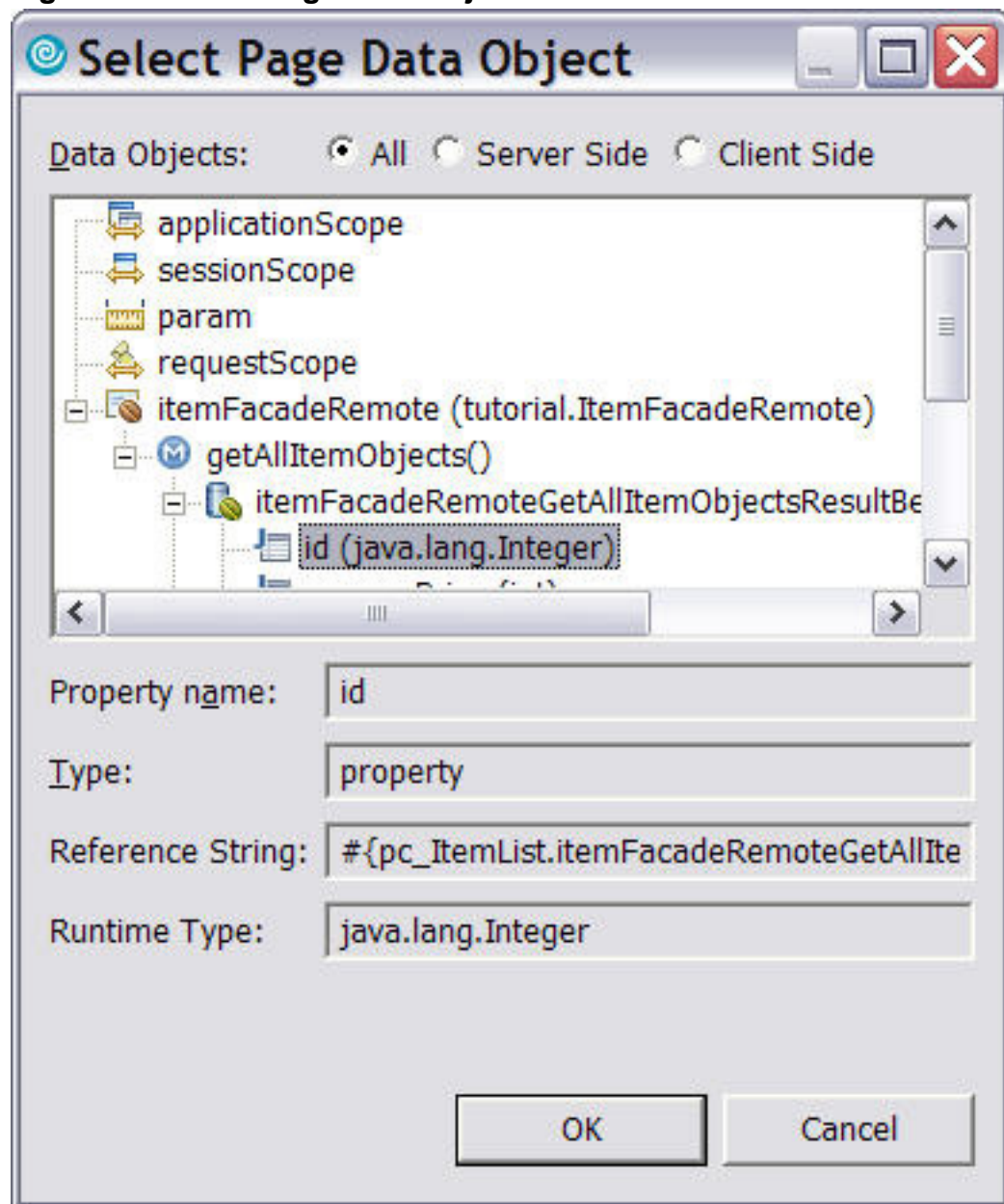
**Figure 22. Configure URL window**



6. Enter `itemBid.jsp` for the URL and `place bid` for the label. Click **OK**.
7. In the Design view, select the **link icon**  next to the link you just created.
8. In the Properties view, click the Parameter tab located just beneath `hx:outputLinkEx`.

9. Click **Add Parameter**, and type `itemId` for the name.
10. Select the **Value field**, and then click the **Select Page Data Object icon**  found in the field. Doing so launches the Select Page Data Object window.
11. Expand `itemFacadeRemote`, then `getAllItemObjects`, and then `itemFacadeRemoteGetAllItemObjectsResultBean`. Select `id` (see Figure 23).

**Figure 23. Select Page Data Object**



12. Click **OK**.
13. Save your work by pressing Ctrl + S.

## Modifying your bid page

Now that your list page is set up, turn your attention to itemBid.jsp. Click the **itemBid.jsp** tab in the editor window. As you did before, delete the default text and replace it with something better, say `Bid On Item`. As you did with itemList.jsp, change the properties of your title so that it's a Heading 1.

Next, it's time to call the Auction session bean:

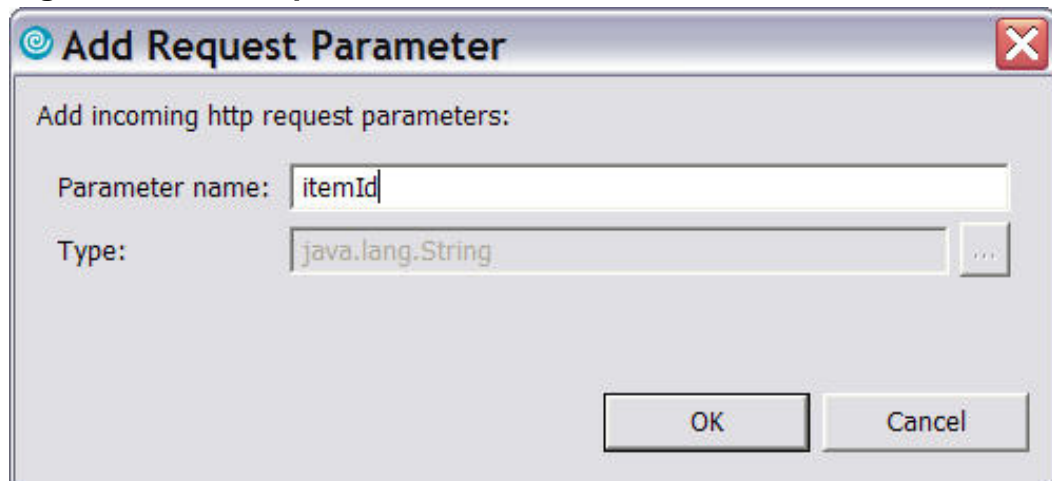
1. Expand the Data drawer in the Palette view.
2. Click **EJB Session Bean**.
3. Click **itemBid.jsp**, below the title, to open the Session Bean window.
4. Click **New EJB Reference** to open the EJB Reference window.
5. Ensure that **Enterprise Beans in the workspace** is selected.
6. Expand RAD tutorialEAR.
7. Expand RAD tutorial.
8. Select **Auction**.
9. Click **Finish**. You are back on the Session Bean window.
10. Verify that the Auction reference is highlighted. Click **Next**.
11. Select **Use default context properties for doing a lookup on this reference**.
12. Click **Next**.
13. From the Select Method window, select **placeBid**. Be sure **Create input form and results display** is selected.
14. Click **Finish**.

As before, you've made a connection to a session bean and Rational Application Developer has done almost all the work. Before you move on, display the parameter

you passed into this page:


1. In the Page Data view, right-click **Scripting Variables**.
2. Select **New > Scripting Variable > Param Scope Variable** to open the Add Request Parameter window (see Figure 24).

**Figure 24. Add Request Parameter window**



3. Enter the name you gave the parameter (`itemId`, in this example).
4. Click **OK**.

Now that you have your variable, display it on your page:

1. Expand the Faces Component drawer in the Palette view.
2. Select **Output**.
3. Click **itemBid.jsp**.
4. With the new component selected in the Design view, click the **Properties** view.
5. Click the **Select Page Data Object**  icon next to the Value field to open the Select Page Data Object window.
6. Expand param.
7. Select **itemId**.

8. Click **OK**.

When your page is called from the item list, the item id is displayed in this field.

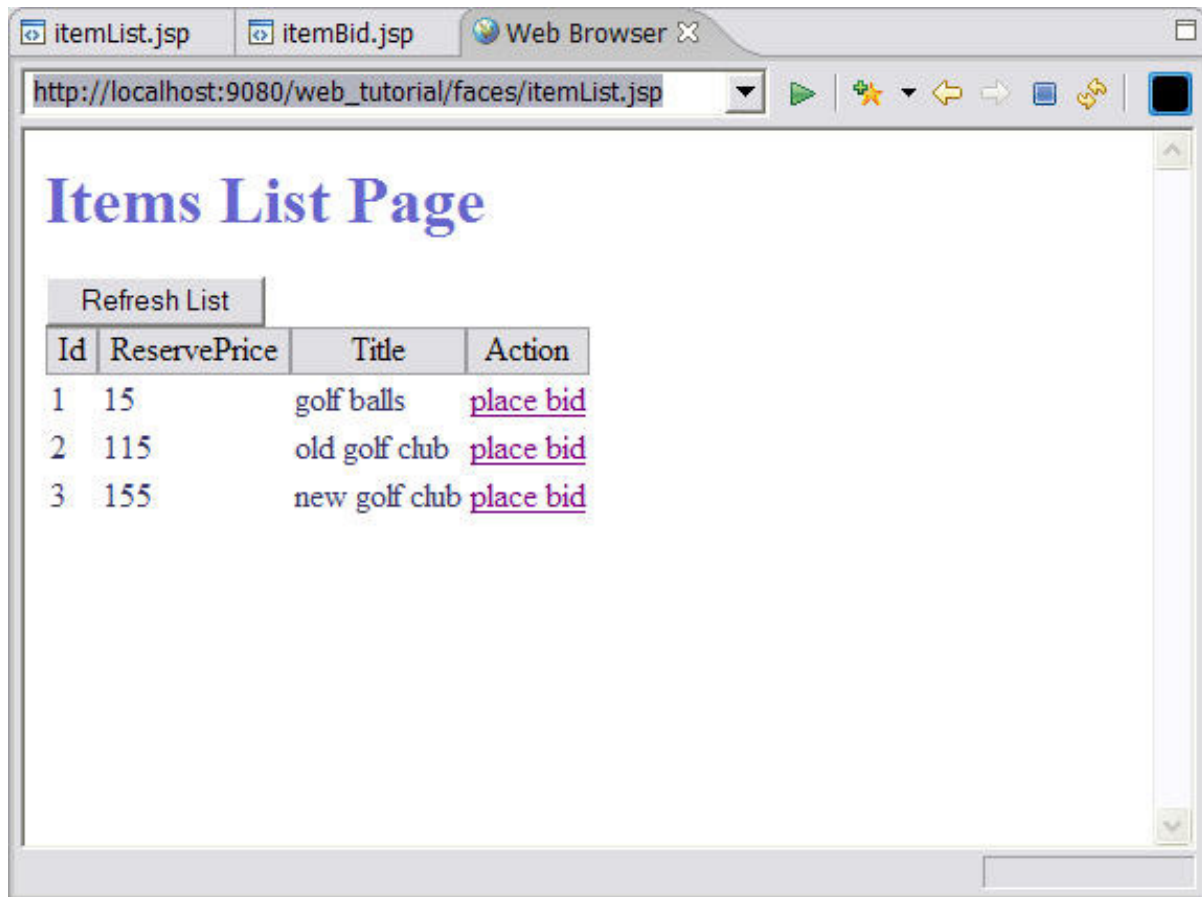
## Testing your Web application

It's time to test your application and see how it works:

1. In the Project Explorer view, expand Enterprise Applications.
2. Right-click **web tutorialEAR**, and **select Run > Run on server**.
3. When your server finishes starting up, expand Dynamic Web Projects.
4. Expand web tutorial.
5. Expand WebContent.
6. Right-click **itemList.jsp**, and select **Run > Run on server**.
7. Click **Finish**.

When your page displays, click **Refresh List**. You should see something like Figure 25. If nothing happens when you click **Refresh List**, stop your server and try running again. If errors appear in the console, you may have missed a step.

### Figure 25. Refreshing the items list



Click the **bid** link for an item. Doing so should take you to the Bid On Item page. Go ahead -- enter the Item Key and a Bid Amount, and click **Submit**. Your page should refresh without any errors. You can use the Universal Test Client or add a call to one of the other methods on the Bid page to verify that the call is working.

---

## Section 5. Summary

In this tutorial, you've extended your Auction application. You added a session facade with a few clicks of your mouse, and you also created an Auction service. You accessed these features with a simple Web application that demonstrated many of the powerful design features that are part of Rational Application Developer. With little coding on your part, you accessed the entity beans created in Part 2. However, you've only scratched the surface of what Rational Application Developer can do.

In the fifth and final part of this series, you'll learn about the role of software testing in

the application development lifecycle. Taking the now-familiar Web Auction application, Part 5 looks at functional testing with the Rational Software Development Platform, using IBM® Rational® PurifyPlus™ to examine performance and memory usage, and takes a brief look at IBM® Rational® TestManager, an automated environment for testing across multiple platforms.

# Resources

## Learn

- [Collating requirements for an application, Part 1](#) is the first part of this tutorial series.
- [A first look at JavaServer Faces, Part 1](#) is helpful if you're just getting started with JSF.
- [JSF for nonbelievers: Clearing the FUD about JSF](#) is good to read if you're intimidated by JavaServer Faces.
- [JSF for nonbelievers: The JSF application lifecycle](#) walks you through the six phases of JSF's request processing lifecycle.
- [IBM® Software Development Platform](#) product page. Find more resources and overview information.
- [The Web's future: XHTML 2.0 and XHTML 1.0 specification](#) are good resources to learn more about XHTML.
- [Creating advanced input forms using JSF and JavaScript with IBM Rational Application Developer V6](#) is great if you are interested in creating forms.
- [Rational® Application Developer V6 Programming Guide](#).
- [developerWorks Rational zone](#) provides many more resources for you.

## Get products and technologies

- [Rational Application Developer Version 6.0](#)

## Discuss

- [Participate in the discussion forum for this content.](#)

## About the author

Nathaniel T. Schutta

Nathaniel T. Schutta, a [Studio B](#) author, is a software developer with extensive experience in the financial services industry, primarily developing Java-based Web applications. A self-proclaimed JUnit evangelist, he has contributed to two corporate Java frameworks, led several study groups, served as a mentor, and developed training materials. A Sun Certified Web Component Developer for the Java 2 Platform Enterprise Edition 1.4, Nathaniel's primary focus is on developing usable, intuitive user interfaces.