

# Build a Web auction client

Skill Level: Introductory

[Michael Donaghy \(mdonaghy@us.ibm.com\)](mailto:mdonaghy@us.ibm.com)

Solutions Developer

IBM

[Colleen Conneaney \(cconnea@us.ibm.com\)](mailto:cconnea@us.ibm.com)

Software Engineer

IBM

13 Dec 2004

This tutorial shows how to use IBM Rational Web Developer for WebSphere Software 6.0 to quickly build a Web auction client that interacts with a database and a set of auction site Web services. You will write minimal amounts of code and instead use Faces Components and SDOs to visually construct your Web pages.

## Section 1. Before you start

### About this tutorial

IBM® Rational® Web Developer for WebSphere® Software 6.0 provides an extensive set of tools designed to quickly build dynamic Web solutions. Rational Web Developer is built on the Eclipse technology, but includes significant productivity enhancements such as enhanced data definition support and rapid application development client tools including JavaServer Faces (JSF) and Service Data Objects (SDO) support.

In this tutorial, you will use Web Developer to build a Web auction client that interacts with a local database and a set of auction site Web services. Your finished Web site will allow you to view your local warehouse stock, view all of your active auction listings, create new auction listings from warehouse stock items, and query the auction site.

You will use the Data perspective to create and deploy an IBM DB2® Universal Database® Express V8.2 database definition. You will use Page Designer and Faces components based on the JSF technology to build Web pages that dynamically access the database and Web services. Using Faces components, you can quickly build dynamic Web pages by dragging user interface components onto your Web page. You can then connect the user interface components to Service Data Objects that represent your data source (such as a database or Web service).

This tutorial is written for developers with minimal Java experience who need to build dynamic Web solutions quickly. Previous experience with visual editors will help you complete the tasks described.

If you are interested in building a similar application, but want to take advantage of Eclipse Web Tools and Java technology, the tutorial [Build a Java auction client: Using the Java Visual Editor in Rational Web Developer 6.0](#) demonstrates how to build a Java® auction client using Rational Web Developer.

For a list of additional information you may find helpful, see [Resources](#).

## Prerequisites

To complete this tutorial you need to have the following software installed:

- [Rational Web Developer 6.0](#) for WebSphere Software.
- [DB2 Universal Database Express V8.2](#)

---

## Section 2. Create a local database

### The local warehouse database

The first step is to create and populate a database of warehouse stock. The warehouse is a simple database consisting of one table, as illustrated below. In the following sections you will create a relational database schema and ddl script for IBM DB2 Universal Database Express V8.2. You will then deploy the ddl to a live database and populate it.

Column Name	Data Type	Column Notes
ID	INTEGER	PK
SHORT_NAME	VARCHAR(80)	
ITEM_DESC	LONG VARCHAR	

## Create the database

Create the database in DB2:

1. Open a DB2 command window: **Start > Programs > IBM DB2 > Command Line Tools > Command Window.**
2. At the prompt, type `db2 create database warehous.`

Use the Rational tool to define the database schema and table:

1. Start Rational Web Developer.
2. Open the Data Perspective: select **Window > Open Perspective > Data.** If the Data perspective is not on the list, use the following steps to enable it:
  1. Select **Window > Preferences.**
  2. Select **Workbench > Capabilities.**
  3. Check **Database Developer.**
  4. Click **OK.**
3. In the upper left, select **Data Definition View.**

## Define the database

Define the database:

1. Create an empty project called WarehouseDB:
  1. Select **File > New > Project...**
  2. Select **Simple > Project**, then click **Next**.
  3. Specify a project name of WarehouseDB, then click **Finish**.
2. Create a database definition called WarehouseDB:
  1. Right-click on the WarehouseDB project and select **New > Database Definition**.
  2. Specify the following properties, then click **Finish**:
    - Folder: WarehouseDB
    - Database name: WarehouseDB
    - Database vendor type: DB2 Universal Database Express V8.2
3. Create a schema called WAREHOUSE:
  1. Right-click on the WarehouseDB database, then select **New > Schema Definition**.
  2. Specify the following properties, then click **Finish**:
    - Folder: / WarehouseDB
    - Database: WAREHOUSEDB
    - Schema Name: WAREHOUSE
4. Create a table definition for WAREHOUSE\_ITEM:
  1. Right-click **WarehouseDB\WAREHOUSEDB\WAREHOUSE**, then select **New > Table Definition...**
  2. Click **Browse...** next to Database/schema.
  3. Select **WarehouseDB > WAREHOUSEDB > WAREHOUSE**, then click **OK**.

4. Specify a name of `WAREHOUSE_ITEM`, then click **Next**.
5. For each column in the `WAREHOUSE_ITEM` table listed on the previous panel, perform the following steps:
  1. Click **Add Another**.
  2. Specify the column name and column type (including string length).
  3. If the column is a primary key (PK), check Key-Column.
  4. Click **Next** after all of the columns have been defined.
  5. Verify that **ID** is the only Column(s) in Primary Key, then click **Next**.
  6. The `WAREHOUSE_ITEM` table has no foreign keys. Click **Finish**.

## Generate the DDL

1. Right-click on **WAREHOUSEDB** (DB2 Universal Database Express V8.2).
2. Select **Generate DDL...**
3. Specify a file name of `WAREHOUSEDB`.
4. Click **Browse** next to the Project field, and select **WarehouseDB**, then click **OK**.
5. Check the following options:
  - Generate SQL DDL with fully qualified names.
  - Generate associated DROP statements.
  - Open SSQL DDL file for editing when done.



6. Click **Finish**.

## Deploy the model

Create a new database connection:

1. In the Database Explorer View, right-click in the white space, then choose **New Connection...**
2. Select **Choose a DB2 Alias**.
3. Specify a name of WarehouseCon, then click **Next**.
4. Specify the following options:

- JDBC Driver: IBM DB2 Application
- Application Alias: WAREHOUS
- The appropriate *user ID* and *password*

**New Database Connection - WarehouseCon**

**Specify connection parameters**  
Select the JDBC driver and required connection parameters.

JDBC driver: IBM DB2 Application

Connection URL details

Alias: WAREHOUS

Comments:

JDBC driver class: COM.ibm.db2.jdbc.app.DB2Driver

Class location: C:\Program Files\IBM\SQLLIB\java\db2java.zip Browse...

Connection URL: jdbc:db2:WAREHOUS

Specify user information

Use your operating system user ID and password

User ID: db2admin

Password: \*\*\*\*\*

Test Connection

< Back Next > Finish Cancel

5. Click **Finish**.
6. When prompted to copy database metadata to a folder, select **No**.

Deploy the model to a live database:

1. Right-click on **WAREHOUSEDDB** (DB2 Universal Database Express V8.2).
2. Select **Deploy...**
3. Keep all items selected, then click **Next**.

4. Keep the defaults on the Data Export Options, then click **Next**.
5. On the Database Connection page, check **Use existing connection WarehouseCon**, then click **Finish**.
6. The first time that an export is run, it is normal for the drop statements to show an error. Click **Commit changes**.

## Populate the database

1. Copy WAREHOUSEDB-populate.sql from the file system into WAREHOUSEDB\Scripts\Data. You can download the SQL script from [Resources](#).
2. Right-click on **WAREHOUSEDB\Scripts\Data\WAREHOUSEDB-populate.sql**.
3. Select **Deploy...**
4. Click **Next**.
5. Select **Commit changes** only upon success and click **Next**.
6. On the Database Connection page, check **Use existing connection WarehouseCon**, then click **Finish**.
7. Click **Commit Changes**.

You now have a local warehouse database populated with stock ready to be listed on the auction site.

---

## Section 3. Connect the Web client to the local database

### Connect to the local database

In this section Faces components are used to build dynamic Web pages that access the local database. You will build two Web pages in this section. One displays a table of all of the stock in the warehouse. The second displays a single stock item in

more detail.

## Create a Dynamic Web Project

Create a Dynamic Web Project to hold all of your application resources:

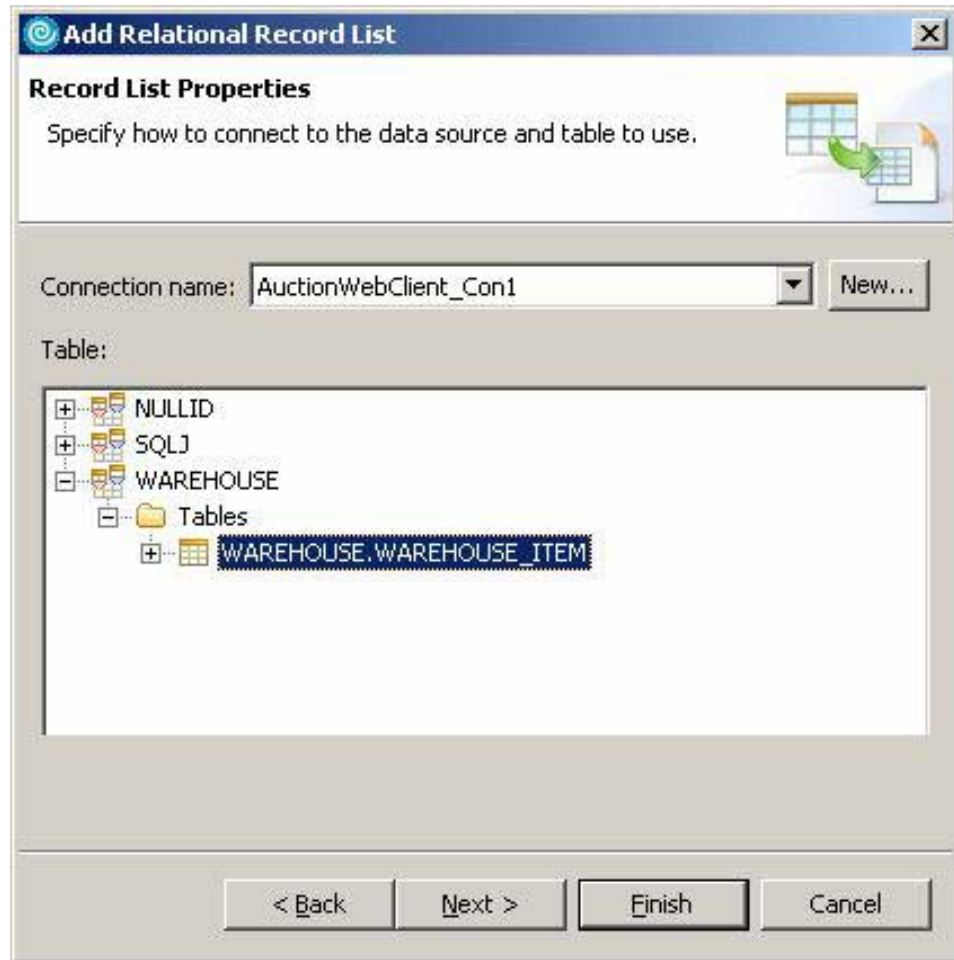
1. First switch to the Web perspective by selecting **Window > Open Perspective > Web**.
2. Select **File > New > Project...**
3. Select **Web > Dynamic Web Project**.
4. Click **Next**.
5. Click **Show Advanced >>**.
6. Specify the following:
  - Name: AuctionWebClient
  - Servlet version: 2.4
  - Target server: WebSphere Application Server v6.0
7. Click **Finish**.

## Create a page to view the warehouse stock

In this section, you will create a relational record list that represents the items in the warehouse database. Then you will populate the relational record list by connecting to the database and selecting the appropriate table. Finally, you will use a data table to display the relational record list on the page.

1. Create the Faces JSP page:
  1. Right-click on **AuctionWebClient\WebContent**.
  2. Select **New > Faces JSP File**.
  3. Specify `showStock.jsp` as the file name.
  4. Click **Finish**.

2. Add a new relational record list:
  1. Delete the default content, **Place content here**.
  2. In the Palette view, click the Data drawer to expand it.
  3. Drag the **Relational Record List** from the **Palette** onto the blank page.
  4. In the Name field, type `all_stocklist`. Make sure that **Add data controls** is checked.
  5. Click **Next**.
  6. Click **New** to create a new database connection.
  7. Under **Choose an existing connection**, select **WarehouseCon**.
  8. Click **Finish**.
  9. Select **Tables > WAREHOUSE.WAREHOUSE\_ITEM**.



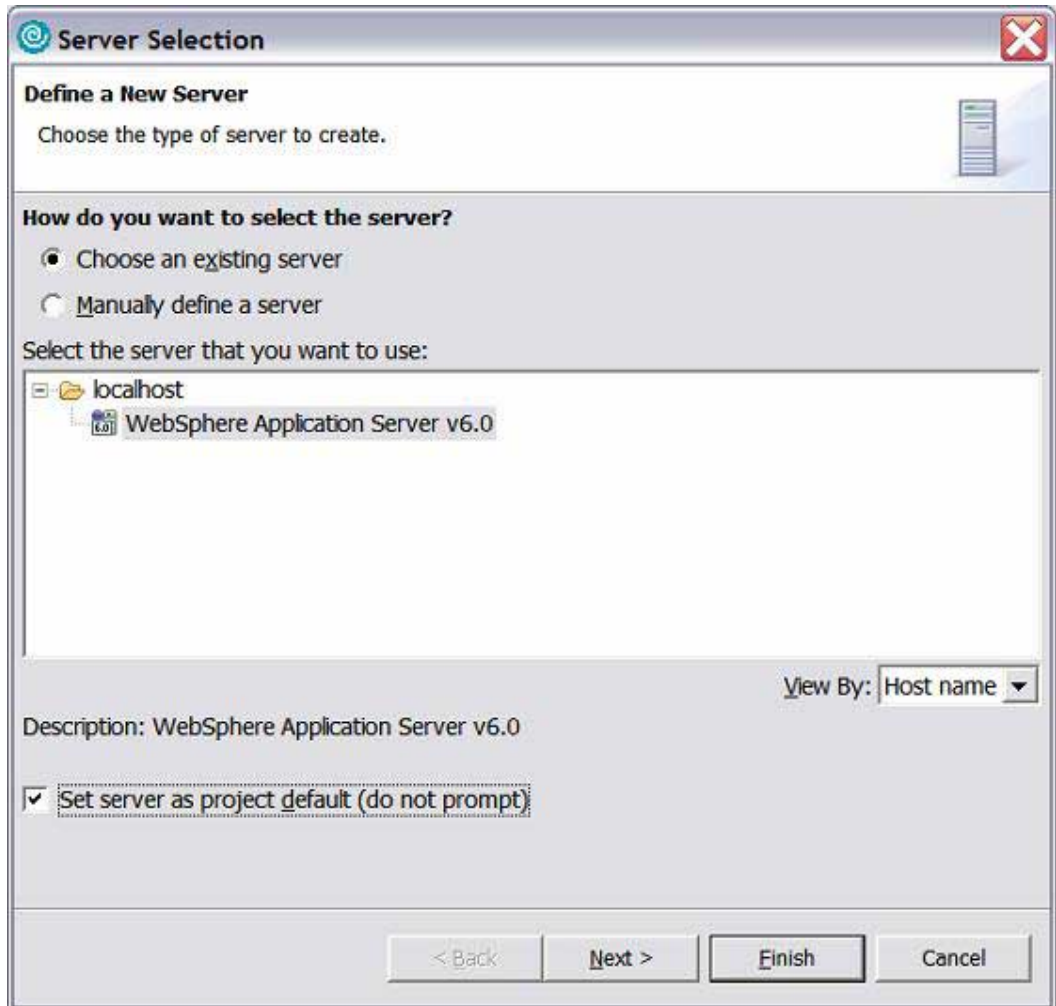
10. Click **Finish**.

## Test the Web site

If you do not have **Build Automatically** checked, build your project before testing:

1. In the Project Explorer view, highlight **AuctionWebClient**.
2. From the menu, select **Project > Build Project**.
3. In the Project Explorer view, right-click **showStock.jsp**.
4. Select **Run > Run on Server...**
5. Under **Select the server that you want to use**, select **WebSphere Application Server v6.0**.

6. Check **Set server as project default (do not prompt)**.



7. Click **Finish**.

## Fine-tune the warehouse data table

When testing the Web site, you probably noticed the data was not sorted and the table was hard to read. Let's go back and enhance the data table:

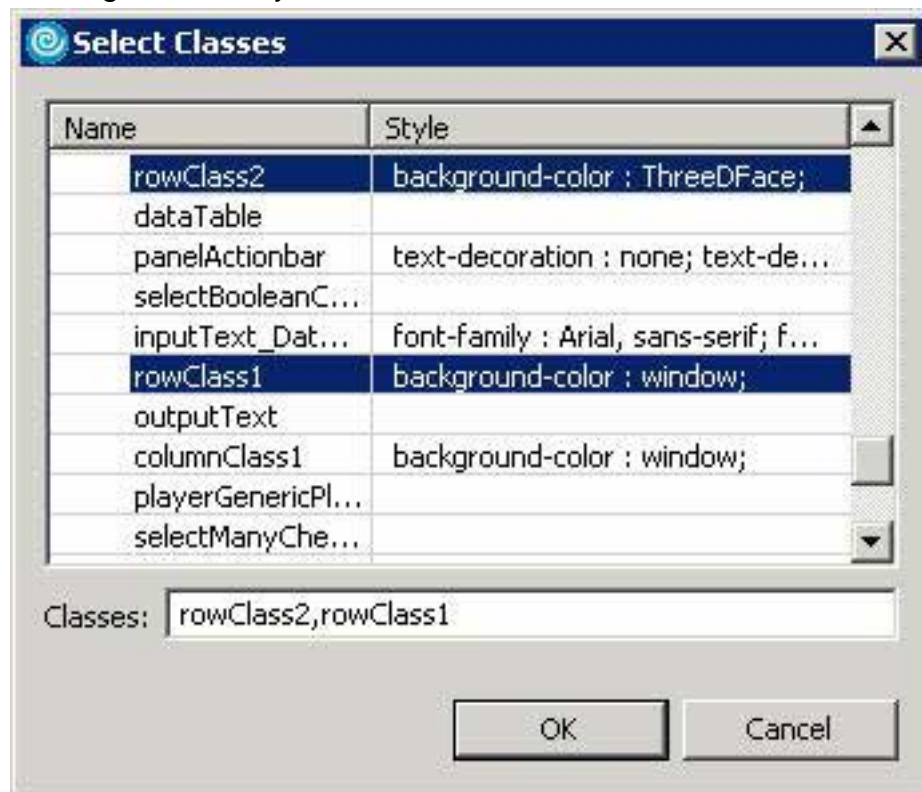
1. Order the data:
  1. In the Page Data view (by default located in the bottom left of the perspective), right-click **all\_stocklist (Service Data Object)**.

2. Select **Configure**.
3. Click the Conditions tab.
4. Click the Order by tab in the bottom half of the window.
5. Under **Available Columns**, highlight **ID** and click the > button. ID is now displayed under Order by on the right side of the window.



6. Click **Close**.
2. Format the data table:
    1. Click anywhere in the data table.
    2. In the Properties view, click **h:dataTable** on the left.
    3. In the right side of the view, click the Item\_desc column under **Label**, then click **Remove**.

4. Click **Id** under **Label** and change it to `Item ID`.
5. Similarly, click **Short\_name** under **Label** and change it to `Title`.
6. Click on **All Attributes** in the upper right of the Properties view.
7. Click on **border** and enter a value of 1.
8. Scroll down and select **rowClasses**.
9. Click on **select classes** (the first of the two buttons in the rowClasses Value field).
10. Expand `/theme/stylesheet.css`.
11. Holding the Ctrl key down, select both **rowClass1** and **rowClass2**.



12. Click **OK**.
  13. Click on **All Attributes** again.
3. Add a pager. Rather than display every record at once, you can use a pager to automatically split the records into pages of a set size without

creating new JSP files:

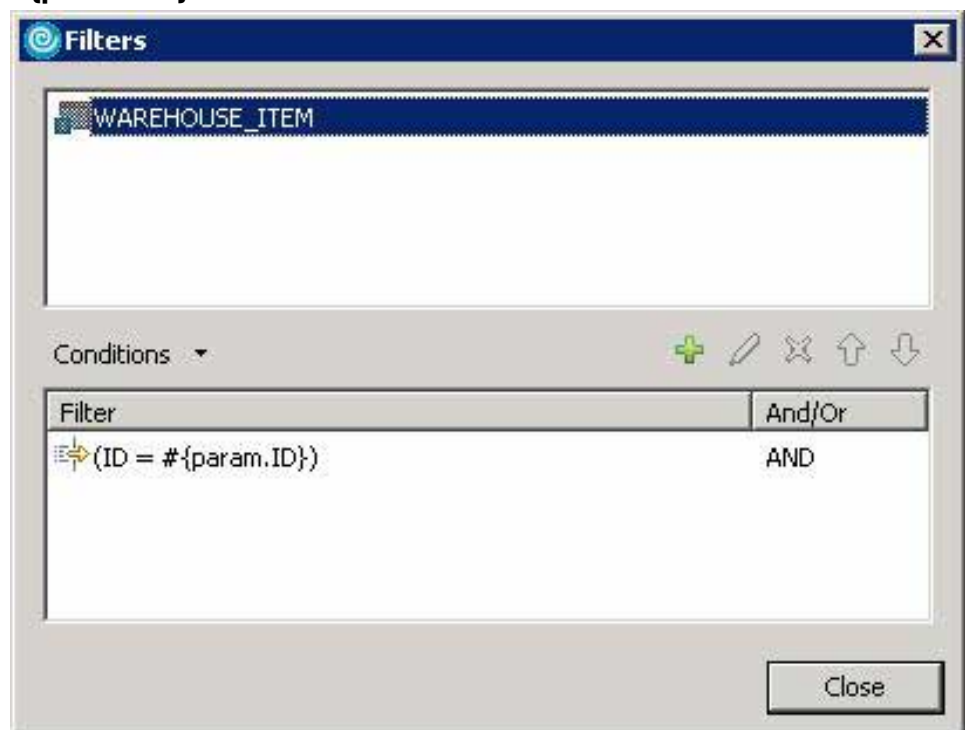
1. Click anywhere in the data table.
  2. In the Properties view, click **h:dataTable** on the left.
  3. In the Properties view, click **Display options** on the left.
  4. In the Rows per page field, enter 10.
  5. Click **Add a Web style pager**.
4. Test the page again to see the results of your formatting.

## Create a page to view an individual warehouse item

Create a relational record that represents a single item in the warehouse database. Use output components to display the relational record. Eventually, you will use this page to create Auction listings from warehouse stock items.

1. Create the file:
  1. Right-click **AuctionWebClient\WebContent**.
  2. Select **New > Faces JSP File**.
  3. Specify `showStockItemDetails.jsp` as the file name.
  4. Click **Finish**.
2. Add a Relational Record:
  1. Delete the default content, **Place content here**.
  2. In the Palette view, click the Data drawer to expand it.
  3. Drag **Relational Record** from the Palette onto the blank page.
  4. In the Name field, type `createauction_record`.
  5. Make sure that the box for **Add input/output controls to display the record on the Web page** is checked.
  6. Under **Create Controls for**, make sure that **Display an existing record (read-only)** is selected.
  7. Click **Next**.

8. In the Table field, select **WAREHOUSE.WAREHOUSE\_ITEM**.
  9. Click **Next**.
3. Filtering the Results. A relational record can only show one database record. Therefore you must filter the database, so only one record appears:
1. Under **Tasks**, click **Filter Results**.
  2. The Filters window opens with the default filter condition, **ID = #{param.ID}**.



3. Click **Close**.
  4. Click **Finish**.
4. Edit the output component labels:
1. Change Short\_name to Title.
  2. Change Item\_desc to Details.

## Use hyperlinks to select stock items to view

Create links on the showStock.jsp page so the user can select a database record to view. `#{param.ID}` represents the ID number of the record that showStockItemDetails.jsp displays. When a link is clicked, the ID of the corresponding record is passed to the showStockItemDetails.jsp page.

1. Open `showStock.jsp`.
2. In the Palette view, click the Faces Components drawer to expand it.
3. Drag the Link component from the Palette directly onto the {ID} output control on the data table. The Configure URL window opens.
4. In the URL field, type `showStockItemDetails.jsp` and leave the Label field empty. Leaving the Label field empty will cause the hyperlink to use the {ID} text as the hyperlink label.
5. Click **OK**.
6. Click the link icon and then click the Parameter tab in the Properties view.
7. Click **Add Parameter**.
8. Type `ID` in the **Name** field for the new parameter.
9. Select the Value field and then click **Select Page Data Object** in the Value field.
10. In the Data Objects field, select the ID column from the `allstock_list` relational record list.
11. Click **OK**.

You will revisit this page further on in the exercise to add the capability to create new auction listings from warehouse stock items. For now, take a moment to test the page:

1. In the Project Explorer view, right-click **AuctionWebClient\showStock.jsp**.
2. Select **Run > Run On Server...**
3. Click a hyperlink in the Item Id column to view an individual warehouse

stock item.

---

## Section 4. Connect the Web client to the auction Web service

### Connect to the auction Web service

In this section you again use Faces components to build dynamic Web pages, this time accessing the auction Web services. You will build three Web pages as well as add functionality to the single warehouse item page from the previous section. The first page displays a table of all of your auctions. The second displays a single auction listing in more detail. You will revisit the single warehouse item page and add the capability to create new auction listings from warehouse stock items. The third page allows you to query the auction listings on the short name field.

Now, create a page to view auctions. In the following steps, use the Web Services Discovery Dialog to find the auction service and create a Web service proxy. Then will add code that will invoke the Web service when the page is loaded. Finally, create output controls to display the results of the Web service call.

1. Create the file:
  1. Right-click **AuctionWebClient\WebContent**.
  2. Select **New > Faces JSP File**.
  3. Specify `showAuctions.jsp` as the file name.
  4. Click **Finish**.
2. Add the Web Service proxy component:
  1. Delete the default content, **Place content here**.
  2. In the Palette view, click the Data drawer to expand it.
  3. Drag the Web Service component from the Palette onto the blank page.
  4. The Web Service Discovery Dialog window opens.



1. Select **Web Services from a known URL**.
2. Enter the URL to the WSDL of the running auction service:  
`http://demo.alphaworks.ibm.com/AuctionSvr/wsdl/com/ibm/ws/AuctionService.wsdl`
3. Click **Go**.
4. Click **Add to Project**. If prompted to enable automatic file overwriting of Web.xml, select **Yes**. The Add Web Service window opens.
5. Add Web Service window:
  1. Under Select a method, select **getAuctions(int,java.lang.String)**.
  2. Uncheck **Create input form and results display (configure details on following pages)**.



3. Click **Finish**.

## Add a Web service call to the page

1. Examine `auctionserviceproxy` in the Page Data view.
  1. Expand `auctionserviceproxy` and `getAuctions(int, java.lang.String)`.
  2. Drag `auctionserviceproxyGetAuctionsResultBean` onto the blank page. The Insert Web Service window opens.
  3. Deselect everything except `status`, `currentPrice`, `shortName`,

**auctionId, endTime.time.**

4. Use the buttons on the right to arrange the selections in the following order: auctionId, shortName, currentPrice, endTime, status.
5. Click **Finish**.
2. Add code to ensure that `getAuctions` is invoked when `showAuctions.jsp` is loaded.
  1. Right-click anywhere in `showAuctions.jsp`.
  2. Select **Edit page code**.
  3. Find the `doAuctionserviceproxyGetAuctionsAction()` method.
  4. Add the following lines just below `try {`. You will get all auctions with a seller ID of **1** and a status of **OPEN**:

```
getAuctionserviceproxyGetAuctionsParamBean().setSellerId(1);
getAuctionserviceproxyGetAuctionsParamBean().setStatus("OPEN");
```

5. Find the `getAuctionserviceproxyGetAuctionsResultBean()` method.
6. Add the following lines immediately before `return auctionserviceproxyGetAuctionsResultBean;`. This code automatically invokes the service if it hasn't already been invoked:

```
if (auctionserviceproxyGetAuctionsResultBean == null)
{
    doAuctionserviceproxyGetAuctionsAction();
}
```

## Format the auction listings page

1. Format the table:
  1. Return to `showAuctions.jsp` and click anywhere on the data table.

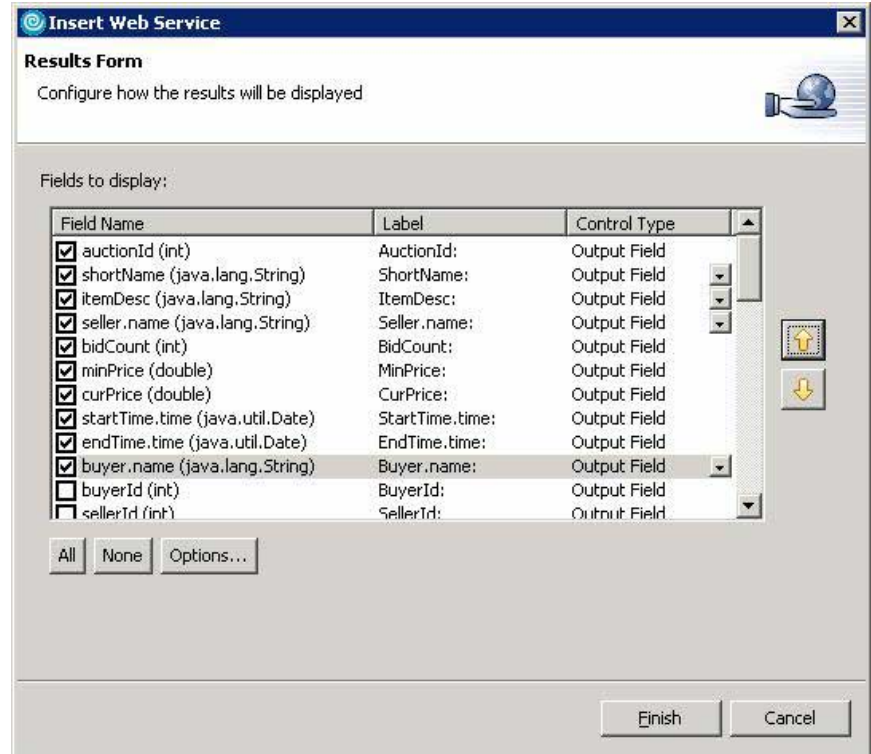
2. In the Properties view, select **h:dataTable** on the left hand side.
  3. In the Columns section to the right, edit the Column labels as follows: AuctionId to `Id`, ShortName to `Item`, CurrentPrice to `Current Price`, Time to `End Time`. To edit a column label, just click the Label entry in the columns table in the Properties view.
  4. Click the All Attributes button in the upper right of the Properties view.
  5. Click **border** and enter a value of 1.
  6. Scroll down and select **rowClasses**.
  7. Click **select classes** (the first of the two buttons in the rowClasses Value field).
  8. Expand **/theme/stylessheet.css**.
  9. Holding the Ctrl key down, select both **rowClass1** and **rowClass2**.
  10. Click **OK**.
  11. Click the All Attributes button again.
2. Format the output components:
    1. Click the {currentPrice} output component on the page.
    2. In the Properties view, change the Format from String to Number.
    3. Change the Type from Decimal to Currency.
    4. Click the {time} output component on the page.
    5. In the Properties view, change the Type from Date only to Date and time.
  3. Take a moment to test the page. In the Project Explorer view, right-click **AuctionWebClient\showAuctions.jsp**. Select **Run > Run On Server...**

## Create a page to view individual auction listings

Create a page to view individual auction listings. The process is very similar to what you did to create the previous page. Add another Web service method to the Web

service proxy created in the previous section, add code to invoke the Web service, and create output controls to display the results of the Web service call.

1. Create the file:
  1. Right-click **AuctionWebClient\WebContent**.
  2. Select **New > Faces JSP File**.
  3. Specify `showAuctionItemDetails.jsp` as the file name.
  4. Click **Finish**.
2. Add the Web Service proxy component:
  1. Delete the default content, **Place content here**.
  2. In the Palette view, click the Data drawer to expand it.
  3. Drag the Web Service component from the Palette onto the blank page. The Add Web Service window opens.
  4. Add Web Service window:
    1. Under Select a method, select **getAuctionInfo(int)**.
    2. Uncheck **Create input form and results display (configure details on following pages)**.
    3. Click **Finish**.
  5. Examine the auctionserviceproxy in the Page Data view:
    1. Expand **auctionserviceproxy** and **getAuctionInfo(int)**.
    2. Drag **auctionserviceproxyGetAuctionInfoResultBean** onto the blank page. The Insert Web Service window opens.
    3. Make sure only the following are selected: **itemDesc**, **bidCount**, **shortName**, **curPrice**, **minPrice**, **auctionId**, **endTime.time**, **buyer.name**, **startTime.time**, and **seller.name**.
    4. Use the buttons on the right to arrange the selections in the following order: **auctionId**, **shortName**, **itemDesc**, **seller.name**, **bidCount**, **minPrice**, **curPrice**, **startTime.time**, **endTime.time**, **buyer.name**.



5. Click **Finish**.

## Add code to invoke the Web service call

Now add code to ensure that `getAuctionInfo` is invoked when `showAuctionItemDetails.jsp` is loaded.

1. Right-click anywhere in `showAuctionItemDetails.jsp`.
2. Select **Edit Page Code**.
3. Find the `doAuctionserviceproxyGetAuctionInfoAction()` method.
4. Add the following lines immediately below `try {`. Create links in the `Id` column of the `showAuctions.jsp` page. Each link will pass the auction ID as a parameter to `showAuctionItemDetails.jsp`. Here you retrieve the value of the ID parameter and use it to get the appropriate `AuctionInfo`:

```
int id = new Integer((String) getRequestParam().get("id")).intValue();
getAuctionserviceproxyGetAuctionInfoParamBean().setAuctionId(id);
```

5. Find the `getAuctionServiceProxyGetAuctionInfoResultBean()` method.
6. Add the following lines immediately before `return auctionServiceProxyGetAuctionInfoResultBean;`. This code automatically invokes the service if it hasn't already been invoked:

```
if (auctionServiceProxyGetAuctionInfoResultBean == null)
{
    doAuctionServiceProxyGetAuctionInfoAction();
}
```

## Format the individual auction listing page

1. Format the results display:
  1. Return to `showAuctionItemDetails.jsp`.
  2. Edit the labels as follows: `AuctionId` to `Id`, `ShortName` to `Item`, `ItemDesc` to `Details`, `Seller.name` to `Seller`, `BidCount` to `Bids`, `MinPrice` to `Minimum Price`, `CurPrice` to `Current Price`, `StartTime.time` to `Start Time`, `EndTime.time` to `End Time`, `Buyer.name` to `Buyer`.
2. Edit the output components:
  1. Click the `{minPrice}` output component on the page.
  2. In the Properties view, change the Format from `String` to `Number`.
  3. Change the Type from `Decimal` to `Currency`.
  4. Repeat steps 1 through 3 for **curPrice**.
  5. Click the **{time}** output component next to `Start Time`.
  6. In the Properties view, change the Type from `Date only` to `Date and time`.
  7. Repeat steps 5 through 6 for the `{time}` output component next to `End Time`.

3. Create links on the showAuctions.jsp page so the user can select an auction listing to view. `#{param.ID}` represents the ID number of the record that showAuctionItemDetails.jsp displays. When a link is clicked, the ID of the corresponding record is passed to the showAuctionItemDetails.jsp page:
  1. Open showAuctions.jsp.
  2. In the Palette view, click the Faces Components drawer to expand it.
  3. Drag the **Link** component from the Palette directly onto the `{auctionId}` output control on the data table. The Configure URL window opens.
  4. In the URL field, type `showAuctionItemDetails.jsp` and leave the Label field empty. Leaving the Label field empty causes the hyperlink to use the `{auctionId}` text as the hyperlink label. Click **OK**.
  5. Click the link icon and then click the Parameter tab in the Properties view.
  6. Click **Add Parameter**.
  7. Type `id` in the Name field for the new parameter.
  8. Select the Value field and then click the Select Page Data Object button in the Value field.
  9. In the Data Objects field, select the `auctionId` field under `auctionserviceproxyGetAuctionsResultBean`.
  10. Click **OK**.
4. Take a moment to test the page. In the Project Explorer view, right-click **AuctionWebClient\showAuctions.jsp**. Select **Run > Run On Server...** Click on a hyperlink in the ID column to view an individual auction.

## Create new auction listings from warehouse items

Add the capability to create auction listings from warehouse stock items. You will revisit the showStockItemDetails.jsp page and add the capability to create new auction listings. Once again, add another method to the Web service proxy. Then add input controls and in the next panel add a submit button to invoke the Web

service call.

1. Open `showStockItemDetails.jsp`.
2. Drag the Web Service component from the Palette onto the page. The Add Web Service window opens.
3. Add Web Service window:
  1. Under Select a method, select **`createAuction(com.ibm.www.Credentials, com.ibm.www.AuctionInfo)`**.
  2. Uncheck **Create input form and results display (configure details on following pages)**.
  3. Click **Finish**.
4. Examine `auctionserviceproxy` in the Page Data view:
  1. Expand **`auctionserviceproxy`** and **`createAuction(com.ibm.www.Credentials, com.ibm.www.AuctionInfo)`**.
  2. Drag **`auctionserviceproxyCreateAuctionParamBean`** onto the blank page. The Insert Web Service window opens.
  3. Make sure only the following are selected: **`myAuth.Email`**, **`myAuth.password`**, and **`auctionInfo.minPrice`**.
  4. Click **Finish**.
5. Format the labels:
  1. Change `MyAuth.Email` to `Email`.
  2. Change `MinPrice` to `Minimum Price`.
6. Change the password input component to an Input - Password field:
  1. Right-click on the password input field.
  2. Select **Delete**.
  3. In the Palette view, click the Faces Components drawer to expand it.
  4. Drag the Input - Password component from the Palette to just

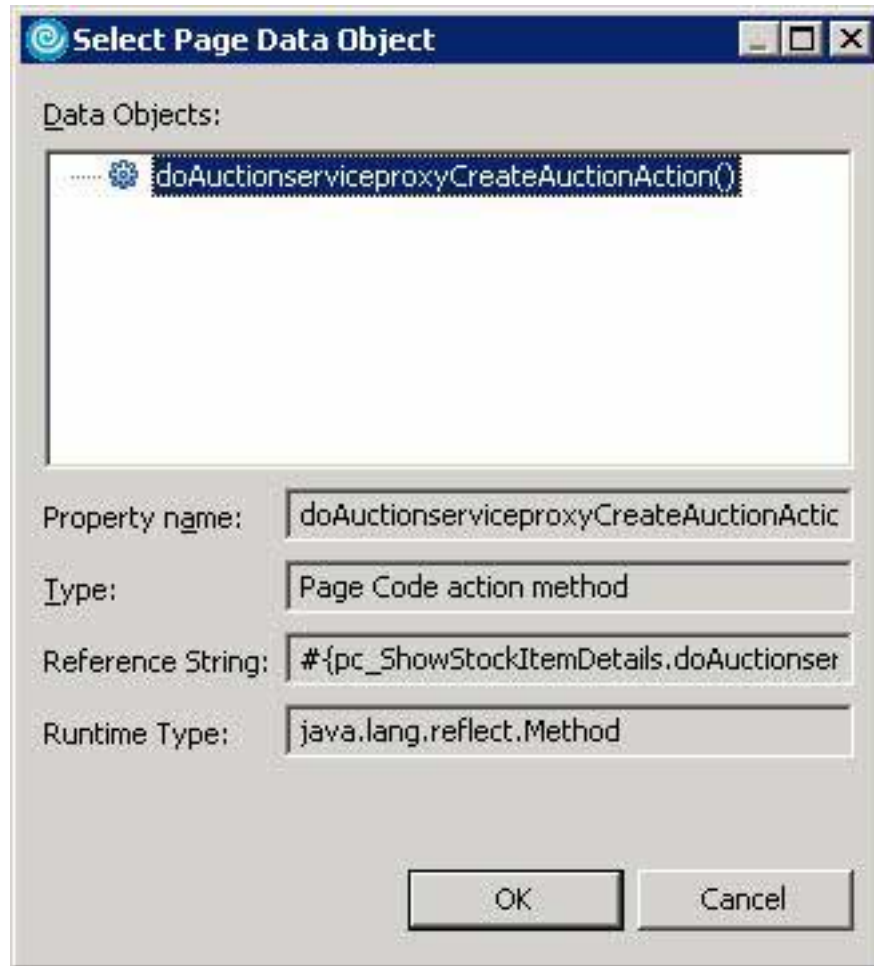
below the EMail input field.

5. In the Page Data view, expand **auctionserviceproxy**, (**com.ibm.www.Credentials**, **com.ibm.www.AuctionInfo**), **auctionserviceproxyCreateAuctionParamBean**, and **myAuth(com.ibm.www.Credentials)**.
6. Drag **password (java.lang.String)** onto the Input - Password field you just created.

## Add an action to the submit button

Now add a submit button to invoke the Web service call.

1. Add an action to the Submit button:
  1. Click **Submit**.
  2. In the Properties view, click **All Attributes** in the upper right of the Properties view.
  3. Click in the Value field next to action.
  4. Click the down arrow and select **Compute...**
  5. The Select Page Data Object window opens. Click **OK**.



6. Click **All Attributes** again.
2. Edit the submit button action code:
  1. Click the Submit button.
  2. Click the Quick Edit tab next to the Properties view.
  3. Replace the existing code with the following code. This code sets the shortName, ItemDesc, and sellerId of the new auctionInfo object with the values from the warehouse stock item. And then creates the auction.

```
try {
    AuctionInfo auctionInfo =
    getAuctionserviceproxyCreateAuctionParamBean().getAuctionInfo();
    Calendar calendar = Calendar.getInstance();
    auctionInfo.setShortName((String)getText2().getValue());
    auctionInfo.setItemDesc((String)getText3().getValue());
```

```
auctionInfo.setSellerId(1);
auctionInfo.setStartTime(calendar);
calendar.add(Calendar.DATE, 3);
auctionInfo.setEndTime(calendar);
auctionserviceproxyCreateAuctionResultBean = getAuctionserviceproxy()
    .createAuction(getAuctionserviceproxyCreateAuctionParamBean().getMyAuth(),
        auctionInfo);

} catch (RemoteException e) {
    logException(e);
}
return null;
```

4. Insert a new line before the `return null;` statement in the code you just pasted.
  5. Right-click on the blank line and select **Insert Snippet > Go To Page**.
  6. In the Edit GoToPage Action window, select **showAuctions.jsp** from the drop down.
  7. Click **OK**.
  8. Right-click anywhere in `showStockItemDetails.jsp` and select **Edit Page Code**.
  9. Right-click anywhere in `ShowStockItemDetails.java` and select **Source > Organize Imports** to add the relevant import statements.
3. Take a moment to test the page. In the Project Explorer view, right-click **AuctionWebClient\showStock.jsp**. Select **Run > Run On Server....** Click on a hyperlink in the Item Id column to select an item to post to an auction listing. Supply the email (`person1@ibm.com`), password (`secreta`) and minimum price. Click **Submit**. You should see the item you just submitted added to the list of active auctions.

## Query the auction Web service

In these steps, create a page to query the Auction Service on the short name field of the auction listings. Add one more method to the Web service proxy and add input and output controls to both invoke and display the results of the Web service call.

1. Create the file:
  1. Right-click **AuctionWebClient\WebContent**.

2. Select **New > Faces JSP File**.
  3. Specify `queryAuctionService.jsp` as the file name.
  4. Click **Finish**.
2. Add the Web Service proxy component:
    1. Delete the default content, **Place content here**.
    2. In the Palette view, click on the Data drawer to expand it.
    3. Drag the Web Service component from the Palette onto the blank page. The Add Web Service window opens.
    4. Add Web Service window:
      1. Under Select a method, select **searchAuctions(java.lang.String)**.
      2. Click **Next**.
      3. Under Label, change ShortNameText to `What are you looking for?`
      4. Click **Finish**.

## Format the query page

1. Format the table. You will format the table very similarly to the table on page `showAuctions.jsp`:
  1. Click anywhere on the data table.
  2. In the Properties view, select **h:dataTable** on the left side.
  3. In the Columns section to the right, remove all columns except: Status, CurrentPrice, ShortName, AuctionId, and Time.
  4. Use the Move Up and Move Down buttons to rearrange the columns in the following order: AuctionId, ShortName, Status, CurrentPrice, Time.
  5. Edit the Column labels as follows: AuctionId to `Id`, ShortName to

Item, CurrentPrice to Current Price, Time to End Time. To edit a column label, just click on the Label entry in the columns table in the Properties view.

6. Click the All Attributes button in the upper right of the Properties view.
  7. Click **border** and enter a value of 1.
  8. Scroll down and select **rowClasses**.
  9. Click **select classes** (the first of the two buttons in the rowClasses Value field).
  10. Expand **/theme/stylessheet.css**.
  11. Holding the Ctrl key down, select both **rowClass1** and **rowClass2**.
  12. Click **OK**.
  13. Click on the All Attributes button again.
2. Format the output components:
    1. Click the {currentPrice} output component on the page.
    2. In the Properties view, change Format from String to Number.
    3. Change the Type from Decimal to Currency.
    4. Click the {time} output component on the page.
    5. In the Properties view, change the Type from Date only to Date and time.
  3. Take a moment to test the page. In the Project Explorer view, right-click **AuctionWebClient\queryAuctionService.jsp**. Select **Run > Run On Server....**

## Create the Web client home page

This section shows you how to create a home page for your Web site and add links to make navigation easier:

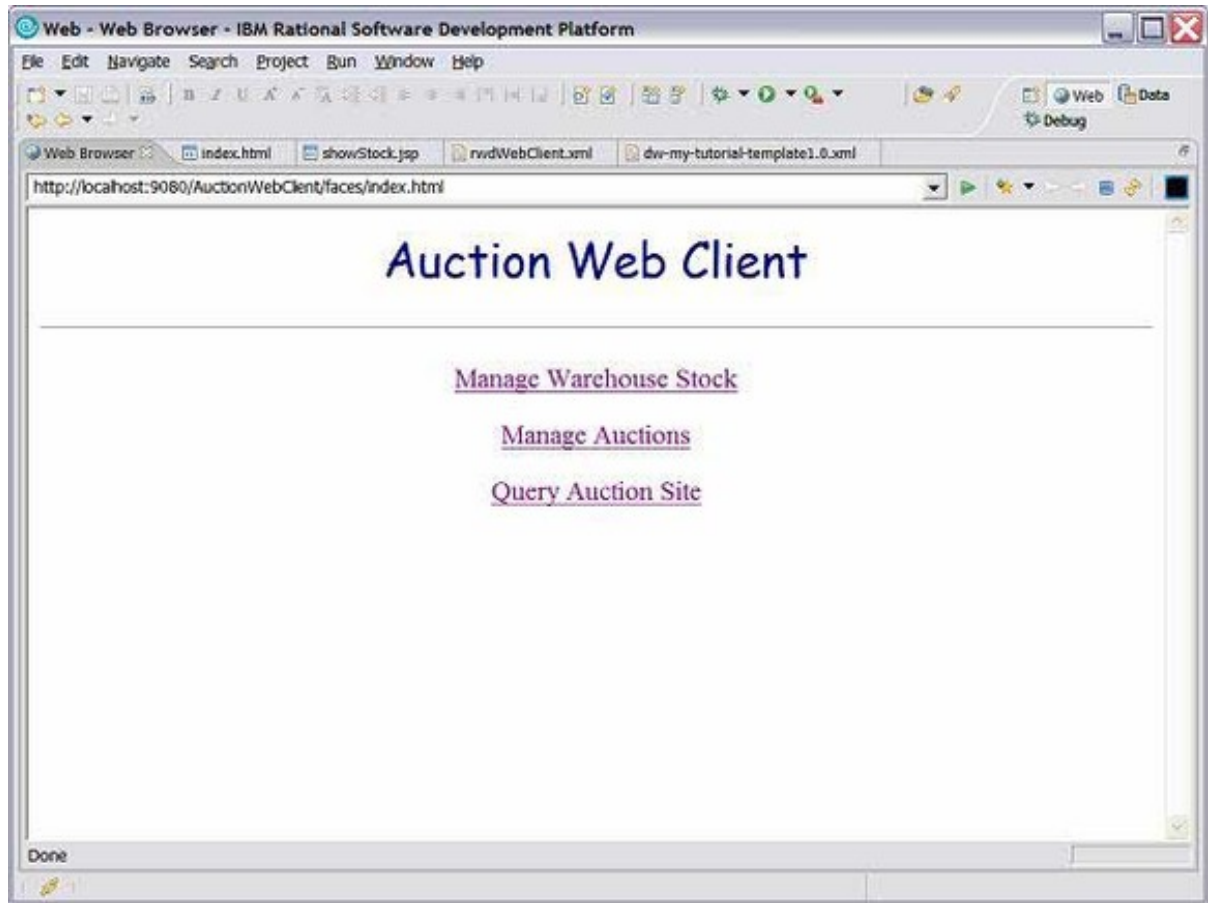
1. Create the file:

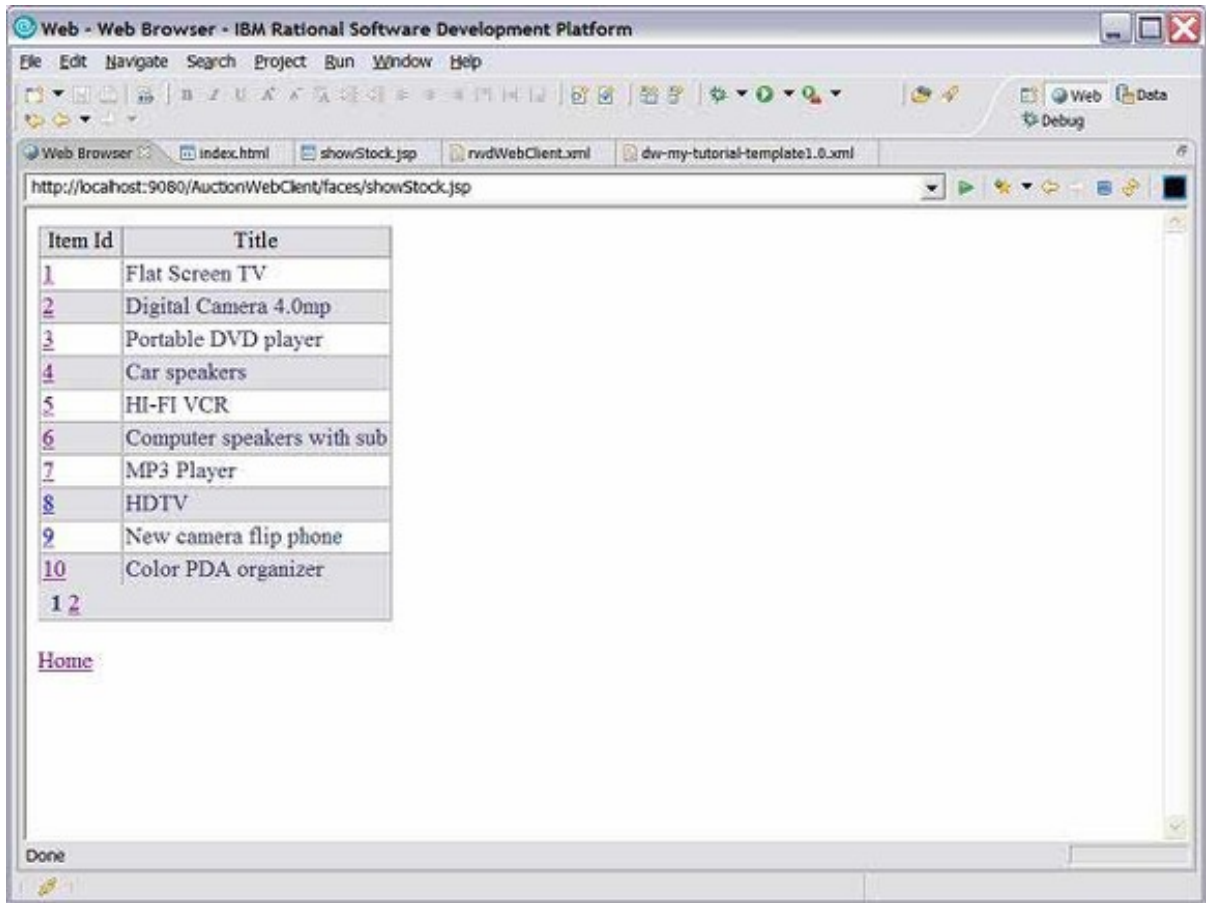
1. Right-click **AuctionWebClient\WebContent**.
  2. Select **New > HTML\XHTML File**.
  3. Specify `index.html` as the file name.
  4. Click **Finish**.
2. Implement the client home page:
1. Delete the default content, **Place content here**.
  2. In its place, type `Auction Web Client`.
  3. Highlight the text, and use the Properties view to format the text. You might want to center align the text, change the font, color, or size, for example:
    1. In the Properties view, click the P tab. Change the Alignment to **Center**.
    2. Click the Text tab. Change the Font to **Comic Sans MS**. Change the color to **Navy**. Use the 'A+' and 'A-' buttons to change the font size.
3. Add a horizontal rule:
1. In the Palette view, click the HTML drawer to expand it.
  2. Drag the Horizontal Rule component from the Palette onto the page below the text.
  3. Use the Properties view to format the horizontal rule as you see fit.
4. Add links:
1. In the Palette view, click the HTML drawer to expand it.
  2. Drag the Link component from the Palette onto the page below the Horizontal Rule. The Insert a Link window opens.
  3. Make sure File is selected.
  4. Click **Browse** and select **Servlet....**
  5. Select **Faces Servlet** and click **OK**.

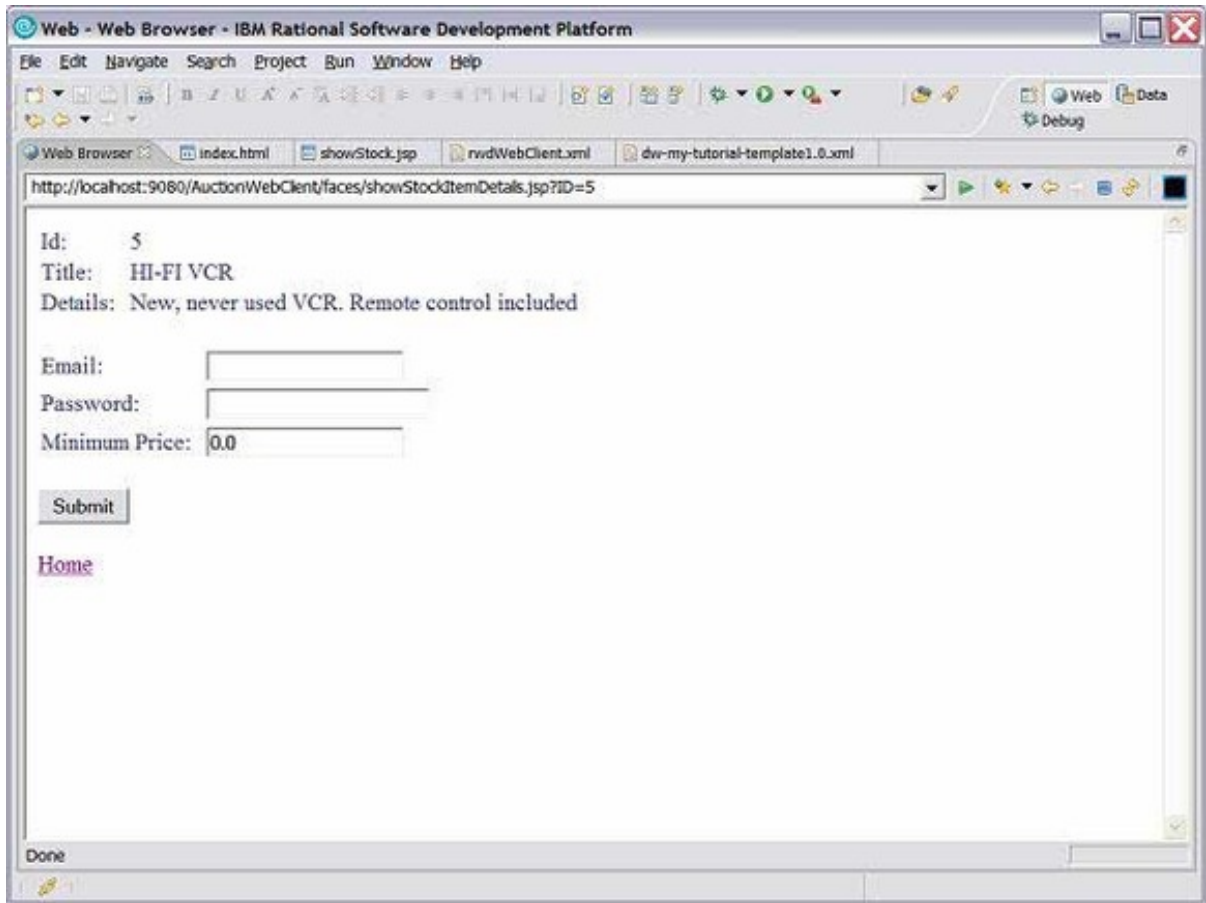
6. In the URTime to End TimeL field, replace the \* with `showStock.jsp`.
  7. Change the Link text to `Manage Warehouse Stock`. Click **OK**.
  8. Highlight the Manage Warehouse Stock link. Use the Properties view to format it as you see fit.
  9. Repeat steps 1 through 8 to create a Manage Auctions link to `showAuctions.jsp` and a Query Auction Site link to `queryAuctionService.jsp`.
5. Finally, add links to the home page from each of your .jsp pages:
1. Open `showStock.jsp`.
  2. In the Palette view, click the HTML drawer to expand it.
  3. Drag the Link component from the Palette onto the page below the Horizontal Rule. The Insert a Link window opens.
  4. Make sure File is selected.
  5. Click **Browse** and select **File....**
  6. Expand **WebContent** and select **index.html**.
  7. Click **OK**.
  8. Change the Link text to `Home`. Click **OK**.
  9. Repeat steps 3 through 8 to create Home links on `showStockItemDetails.jsp`, `showAuctions.jsp`, `showAuctionItemDetails.jsp`, and `queryAuctionService.jsp`.

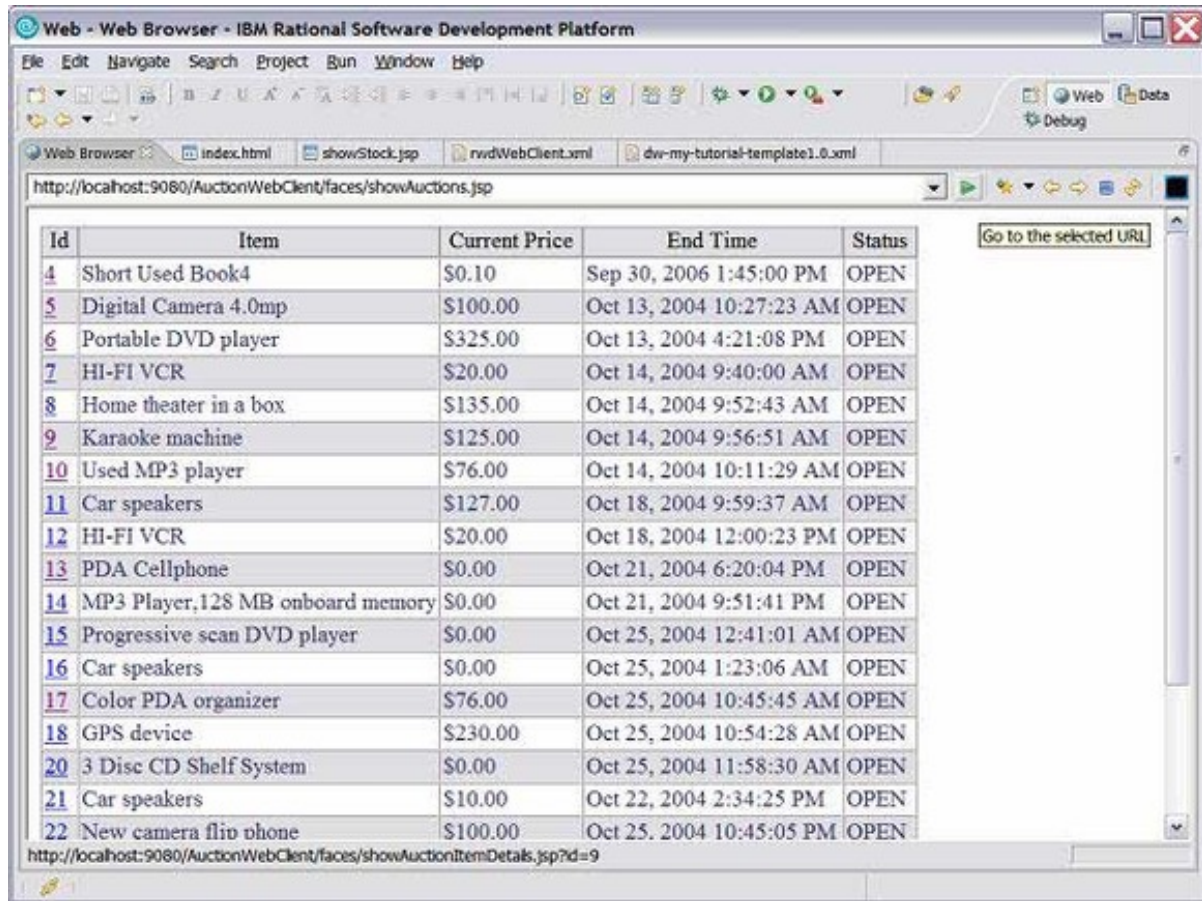
## Test the Web client

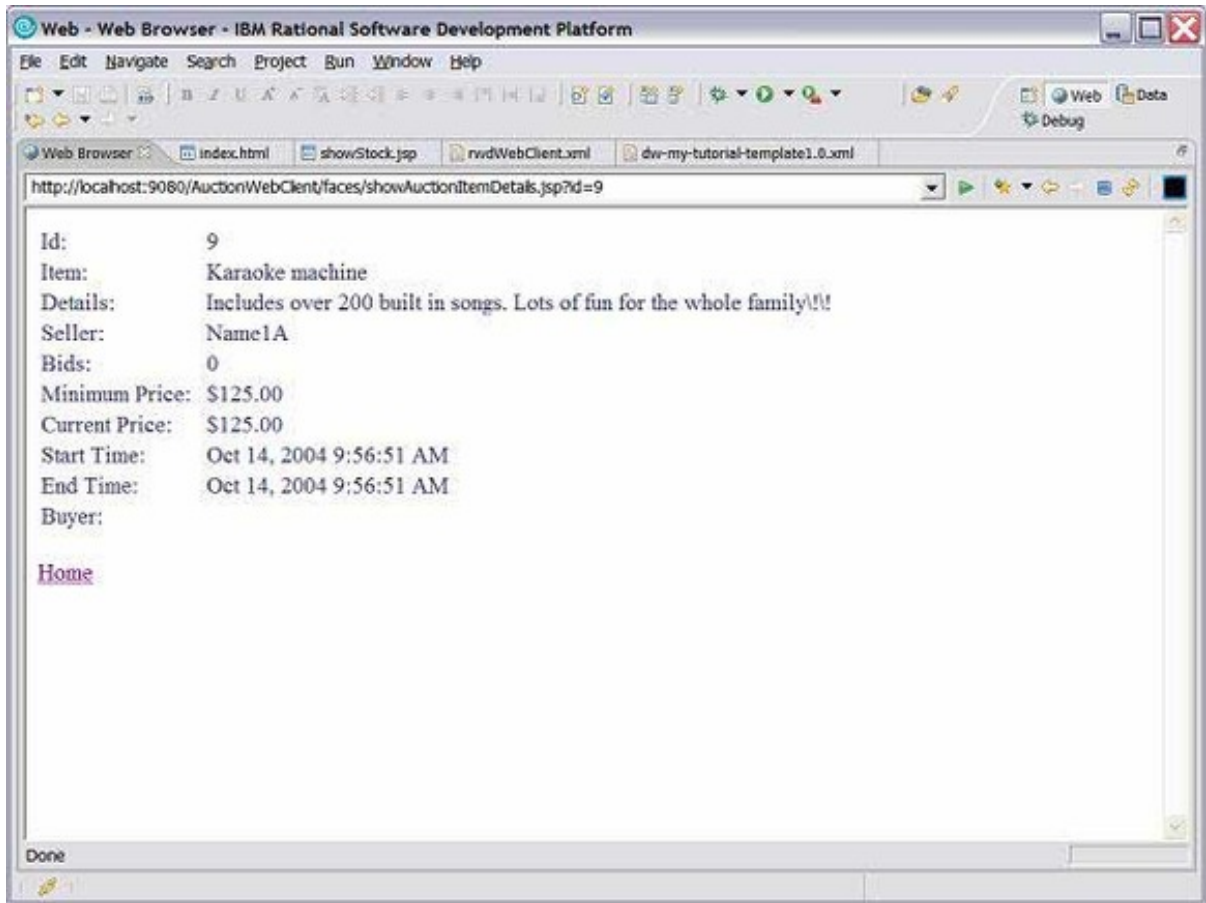
Finally, test the Web client as a whole. Your pages should look similar to the following:

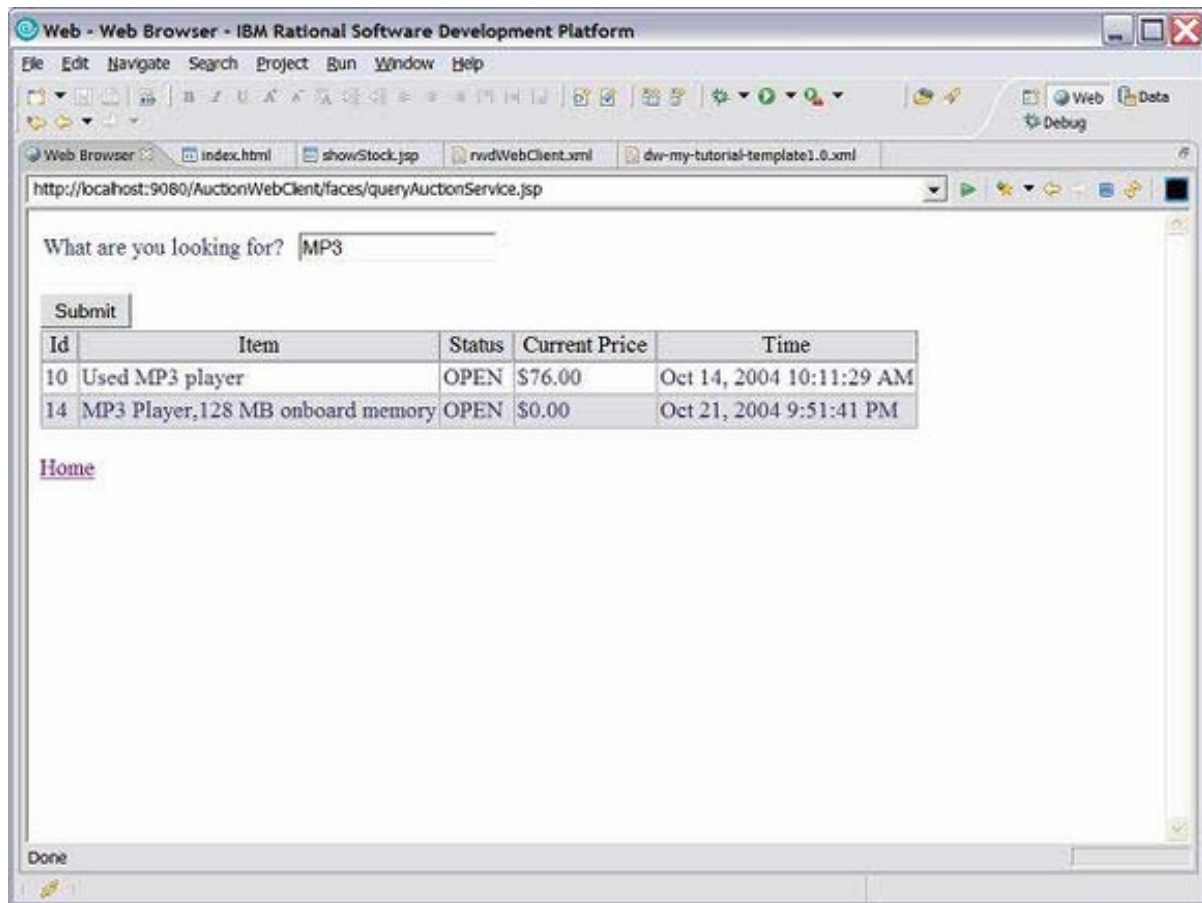












---

## Section 5. Summary

In this tutorial you've seen how to use IBM Rational Web Developer 6.0 to build dynamic Web pages quickly. Using Page Designer, you dragged and dropped Faces Components to build user interfaces. Then you connected the user interfaces to data sources using SDOs. Your end product is a Web application that, among other things, allows you to view a local database of warehouse stock and create new auction listings from that stock using Web services. You accomplished all of this very quickly and without a lot of manual coding in contrast to the [Build a Web-based client with the Eclipse Web Tools Platform tutorial](#).

# Resources

## Learn

- Refer to the tutorial, [Build a Web-based client with the Eclipse Web Tools Platform](#) for information on how to develop the same scenario using the Eclipse Web Tools Platform.
- Refer to the tutorial, [Build a Web service using the Eclipse Web Tools Platform](#) for information on how to develop the Web service used in this scenario.
- To try out a working version of the auction application, go to the [Auction Web Service and Web Client Demo](#) page and click View Demo. The auction application is hosted on [alphaWorks](#).
- The IBM WebSphere Developer Technical Journal has a five part series on [Developing JSF Applications using WebSphere Studio V5.1.1](#).
- For more on developing JavaServer Faces UIs, see the tutorial, [UI Development with Java Server Faces](#).

## Get products and technologies

- Download [sqlScript.zip](#), the sql script for populating the local database used in this tutorial.

## Discuss

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

## About the authors

### Michael Donaghy

Michael Donaghy is a Solutions Developer with the [IBM Solutions Builder Express Portfolio](#) Development in RTP, NC. He has developed both enterprise and mid-market solutions for IBM. He holds a Master of Computer Science from North Carolina State University. In his spare time, he enjoys water skiing.

---

### Colleen Connearney

Colleen Connearney is a Software Engineer with the IBM SMB Development Center in RTP, NC. In the three years she's been at IBM, Colleen has worked as a Java and Eclipse plug-in developer with the

Integrated Runtime team. In her spare time she enjoys playing ultimate frisbee.