

Build a database-enabled auction Web service

Skill Level: Intermediate

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

23 Nov 2004

This tutorial shows you how to build a Web service using the Eclipse Web Tools Platform. The Web service interacts with a Cloudscape (Apache Derby) database and is deployed to Apache Tomcat.

Section 1. Before you start

About this tutorial

Building applications using the Java™ 2 Enterprise Edition architecture can be an intimidating process. The Eclipse Web Tools Platform provides a base set of tools to make developing in this environment easier.

This tutorial shows you how to create a simple Web service using IBM's Web Tools Platform contribution to the Eclipse project. You will build a simple auction Web service that interacts with a Cloudscape (Apache Derby) database. The resulting Web service is then deployed to Apache Tomcat 4.1. As an alternative, this tutorial also includes instructions for migrating the example to an IBM DB2® Universal Database Express V8.2 and IBM WebSphere® Application Server Express 5.1 environment.

The scope of this tutorial is limited to listing items for auction and querying item records from the auction server. Implementing other functions, such as user registration and auction bidding, is left as an exercise for the user.

Prerequisites

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

- [Eclipse V3.0](#)
- The [IBM contribution to the Eclipse Web Tools Platform](#) and all of its prerequisites
- [Apache Tomcat V4.1.30](#)
- [IBM Cloudscape V10.0](#) or [IBM DB2 Universal Database V8.2 Express](#)

You should also download the attached file, which includes scripts and source code used in this tutorial.

The tutorial is intended for users who have some basic Java programming skills. Source files are included for the auction service, with some implementation left to the user as a separate exercise.

Section 2. Building a database

The database schema

The back end for your simple auction server will be powered by the following two database tables. In a production auction application, a more complex schema including additional tables (for example, a record of bids) would likely need to be included.

Table 1. PERSON

Column name	Data type	Column notes
PERSON_ID	INTEGER	PK, AUTO-INCREMENT
PERSON_NAME	VARCHAR(40)	
ADDRESS1	VARCHAR(80)	
ADDRESS2	VARCHAR(80)	NULLABLE
CITY	VARCHAR(40)	
STATE	VARCHAR(2)	
ZIP	VARCHAR(10)	
EMAIL	VARCHAR(40)	
PASSWORD	VARCHAR(20)	
NICKNAME	VARCHAR(20)	

Table 2. AUCTION_ITEM

Column name	Data type	Column notes
AUCTION_ID	INTEGER	PK, AUTO-INCREMENT
START_TIME	TIMESTAMP	
END_TIME	TIMESTAMP	
MIN_PRICE	DECIMAL(12,2)	
CUR_PRICE	DECIMAL(12,2)	
BID_CNT	INTEGER	
SELLER_ID	INTEGER	FK
BUYER_ID	INTEGER	FK
SHORT_NAME	VARCHAR(80)	
ITEM_DESC	VARCHAR(32000)	

Create the database tables

The `auctionsvr-create.ddl` script included in the download file will create the necessary tables, the `auctionsvr-populate.sql` script will populate the tables with sample data, and the `auctionsvr-drop.ddl` script will drop tables created for this article.

For IBM Cloudscape V10 environments:

1. In a command window, do the following:
 - a. Change the directory to <Cloudscape V10.0 install directory>\frameworks\embedded\bin
 - b. Run `setEmbeddedCP.bat`
 - c. Run `ij.bat`
2. At the `ij>` prompt, type:
 - a. `connect`
`'jdbc:derby:c:\path\AUCTIONDB;create=true';`
 (where `c:\path\` is where you want to create the database)
 - b. `run 'c:\path\auctionsvr-create.ddl';`

(where `c:\path\` is the location of the `auctionsvr-create` script)

- c. `run 'c:\path\auctionsvr-populate.sql';`
(where `c:\path\` is the location of the `auctionsvr-populate.sql` script)

Alternatively, for DB2 environments:

Open a DB2 command window and execute the following commands:

1. `db2 create database auct_svr`
 2. `db2 connect to auct_svr`
 3. `db2 -tvf c:\path\auctionsvr-create.ddl`
(where `c:\path\` is the location of the `auctionsvr-create.ddl` script)
 4. `db2 -tvf c:\path\auctionsvr-populate.sql`
(where `c:\path\` is the location of the `auctionsvr-populate.sql` script)
-

Section 3. Preparing to build a Web service

Overview

There are two paths you can take when creating a Web service. One path starts with a WSDL file and generates Java skeleton source, and the other starts with Java source and generates a WSDL. This tutorial discusses the second path, starting with the Java source and generating the WSDL.

In a nutshell (with detailed instructions to follow), you will:

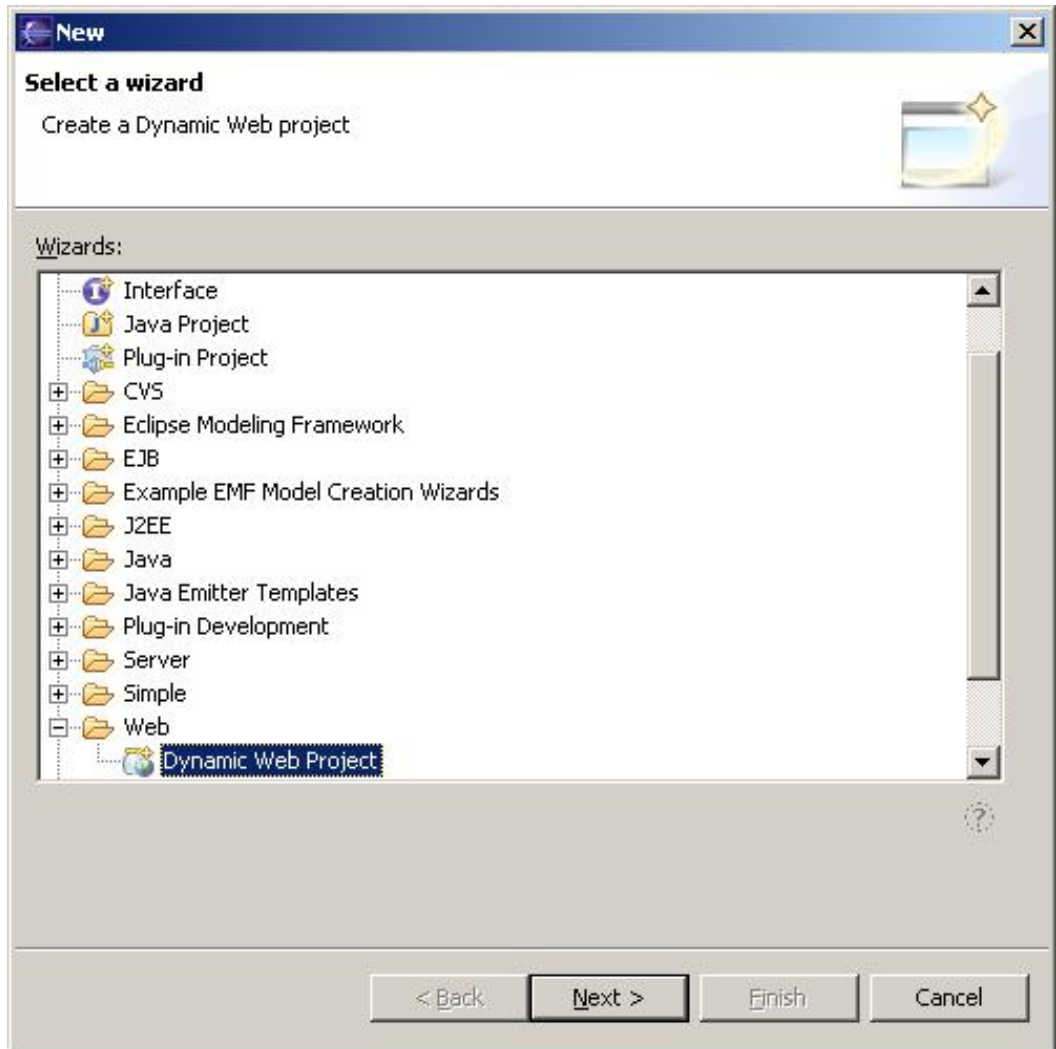
1. Start Eclipse and configure it for Tomcat V4.1
2. Create a new Dynamic Web Project (which will eventually host the Web service)
3. Create a Java source file with public methods that provide some service
4. Create a Web service from the .java files

5. Run the Web Services Explorer or create a Web Service Client

The next section shows how to create a simple auction Web service using Eclipse with the Web Tools Platform contribution. IBM Rational® Web Developer for WebSphere Software V6.0 can be used as an alternative to Eclipse with Web Tools Platform. The detailed instructions explain the method of returning a list of auctions, including a few properties of each auction, in which a seller has sold items.

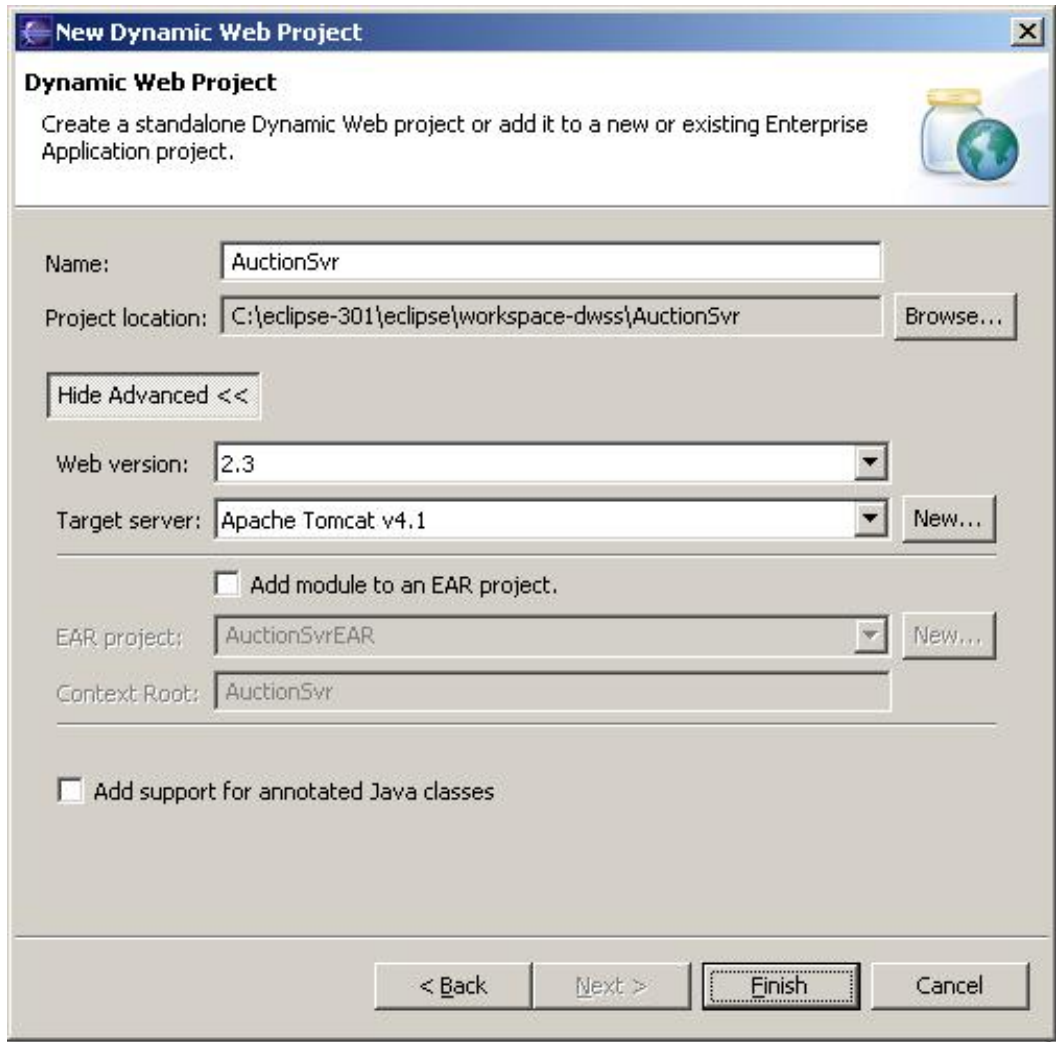
Configure Eclipse for Tomcat V4.1

1. Start Eclipse.
2. Select **Window > Preferences**.
3. Select **Server > Installed Runtimes**.
4. Select **Add...**
5. Select **Apache > Apache Tomcat V4.1**, then click **Next**.
6. Specify properties (sample properties are listed below; your values may be different), then click **Finish**.
Name: **Apache Tomcat V4.1**
Tomcat installation directory: **C:\Tomcat4130**
JRE: **j2re1.4.2_05**
7. The Preferences dialog should now look like this:
Figure 1. Preferences dialog



3. Specify the following, then click **Finish**.
Name: **AuctionSvr**
Web version: **2.3**
Target Server: **Apache Tomcat v4.1**
Add module to an EAR project: **Deselect**

Figure 3. Dynamic Web Project



Overview of source code for this tutorial

The complete source for the examples used in this tutorial (see the Download section) includes several files in the `com.ibm.sample.auctionsrvr.ws` package. (The next section shows how to import these.)

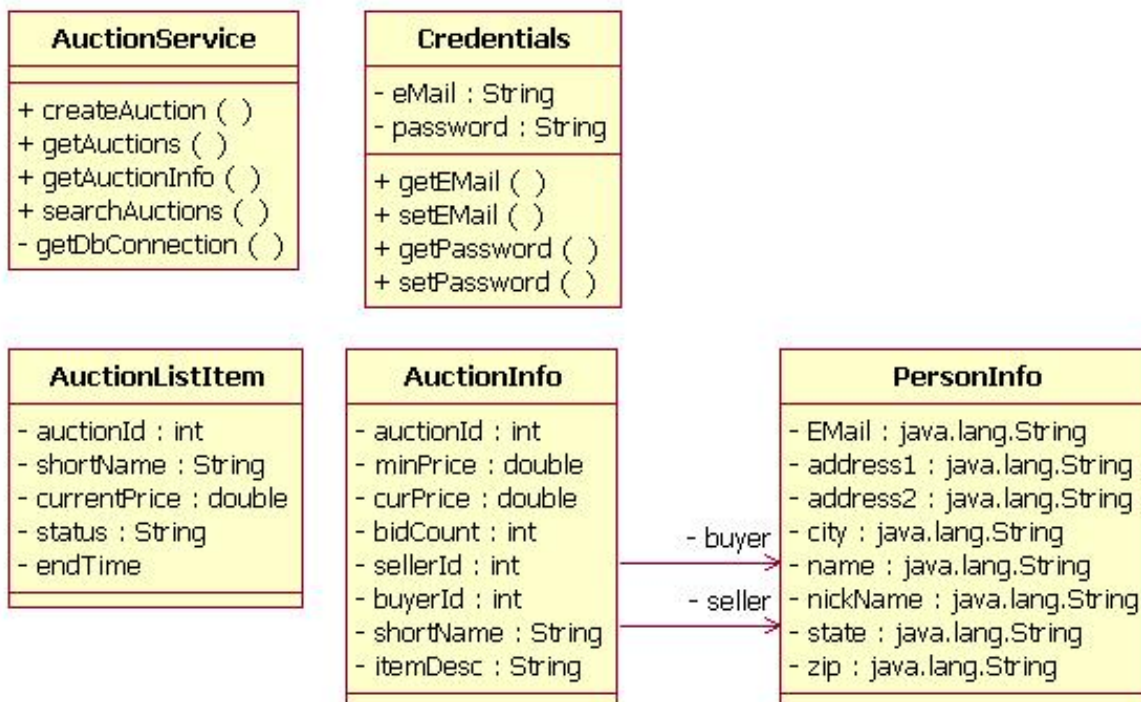
The included classes are:

- **AuctionInfo:** Bean with properties `auctionId`, `startTime`, `endTime`, `minPrice`, `curPrice`, `bidCount`, `sellerId`, `seller`, `buyerId`, `buyer shortName`, and `itemDesc`
- **AuctionListItem:** Bean with the following properties: `auctionId`, `shortName`, `currentPrice`, `endTime`, and `status`

- Credentials: Bean with the following properties: eMail and password
- PersonInfo: Bean with the following properties: eMail, name, nickname, address1, address2, city, state, and zip
- AuctionService: Bean with the following methods:
 - createAuction(Credentials, AuctionInfo): Creates a new auction
 - getAuctions(sellerId, auctionStatus): Returns a list of auctions (AuctionListItem) for a particular seller bases on status (OPEN, CLOSED, ALL)
 - getAuctionInfo(auctionId): Returns a detailed description of an auction (AuctionInfo)
 - searchAuctions(shortDescription): Returns a list of auctions (AuctionListItem) that contain the specified search string

The methods in the AuctionService class will be invoked by the Web service framework. The methods' parameters and return types can be simple values such as integers and strings or datatypes with multiple properties.

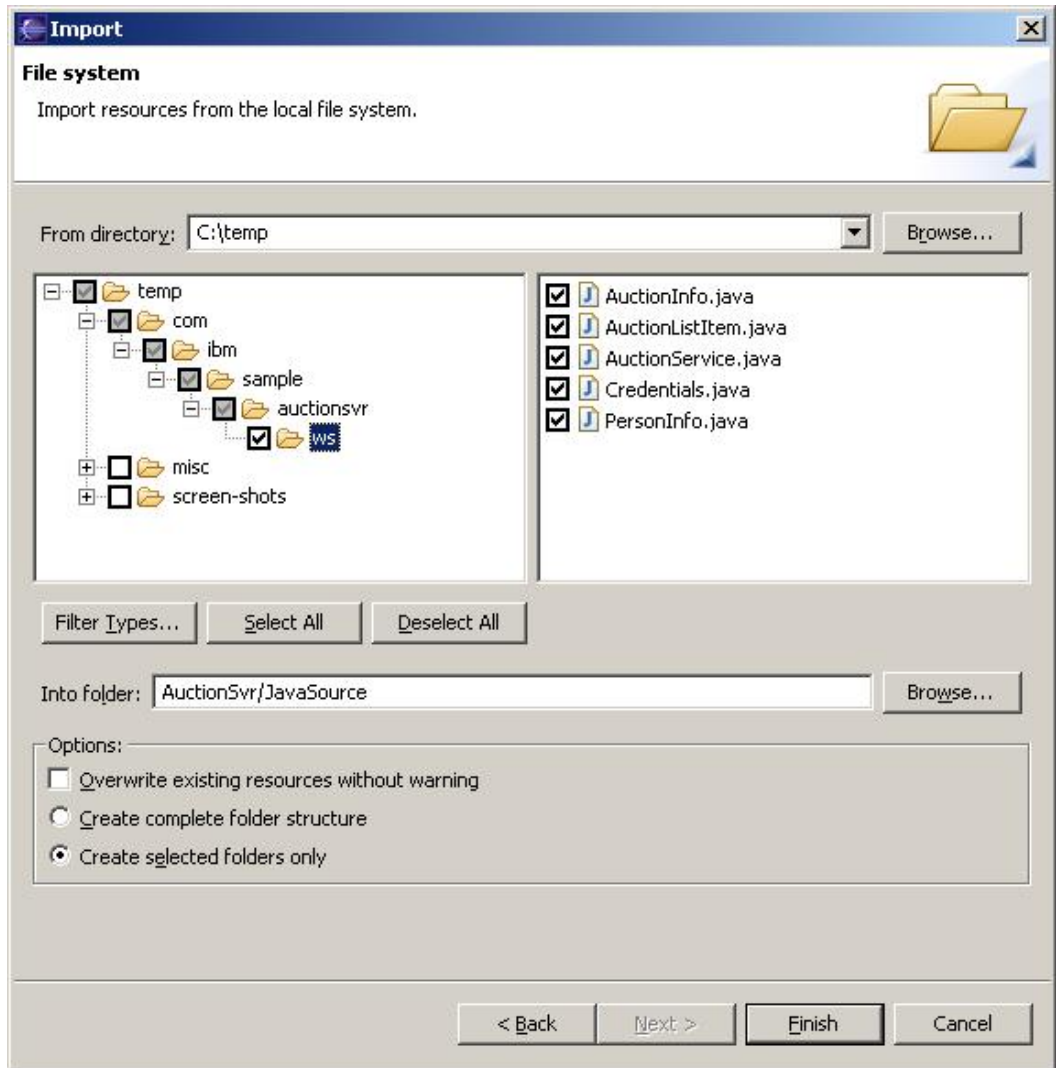
Figure 4. Methods in the AuctionService class



Import the source code

1. Extract the attached source code (see the Download section) in to a directory such as **C:\temp**.
2. Switch to the Java Perspective in Eclipse (**Window > Open Perspective > Other... > Java**).
3. In Eclipse, right-click the **Java Source** folder of the **AuctionSvr** project, and select **Import**.
4. Choose **File system**, then **Next**.
5. In the **From** directory, specify the location you unzipped the files to, such as **C:\temp**. If you typed the directory in without using the **Browse** button, hit the **tab** key to update the left selection window.
6. Select the **com/ibm/sample/auctionsrv/ws** directory, then click **Finish**.

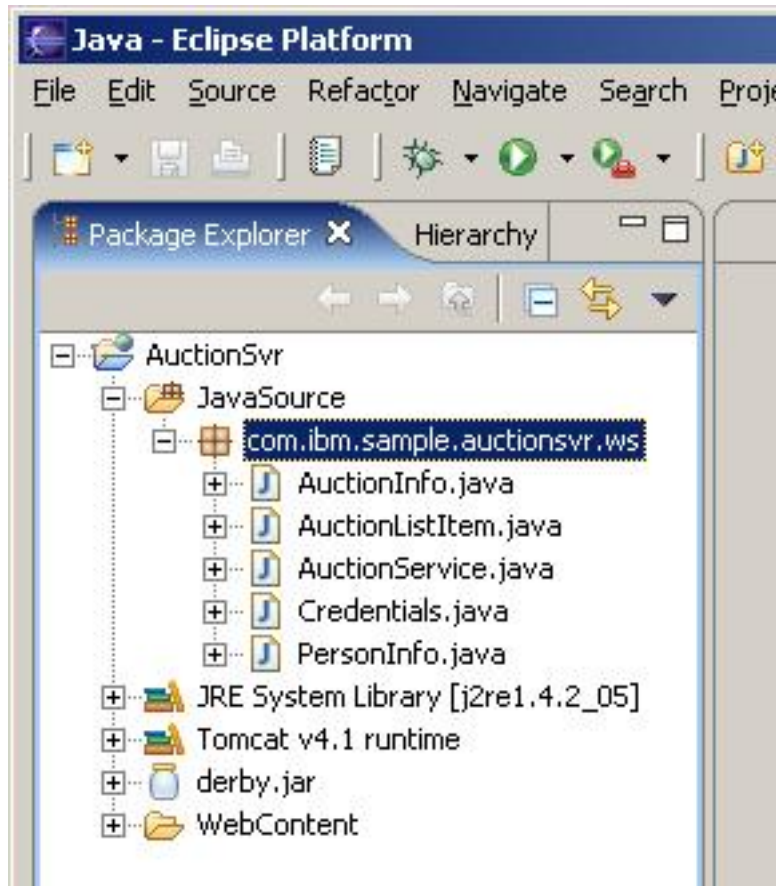
Figure 5. Importing source code



7. From IBM Cloudscape V10, import the **derby.jar** file to **WebContent/lib**. **derby.jar** should automatically be added to the project's classpath. If you are using DB2 instead of Cloudscape 10, use **db2jcc.jar** instead of **derby.jar**.

The Package Explorer View should now look like this:

Figure 6. Package Explorer View



About the database connection method

The sample uses these static variables and the method below to acquire database connections. Note that there are alternatives to the method below, such as JNDI. The developerWorks article "Integrating Cloudscape and Tomcat" (see [Resources](#)) explains how to use JNDI with Tomcat and IBM Cloudscape V5.1. The same general concepts apply to IBM Cloudscape V10.

```
private String JDBC_DRIVER_CLASS = \
"org.apache.derby.jdbc.EmbeddedDriver";
private String AUCTION_JDBC_URL = \
"jdbc:derby:c:\\auctiondb;create=true";

private Connection getDbConnection() {
    Connection connection;
    try {
        Class.forName(JDBC_DRIVER_CLASS);
        connection = DriverManager.get\
Connection(AUCTION_JDBC_URL);
    } catch (ClassNotFoundException e) {
        connection = null;
        e.printStackTrace();
    } catch (SQLException e) {
        connection = null;
    }
}
```

```
        e.printStackTrace();
    }
    return connection;
}
```

When using DB2, modify the lines containing the JDBC driver class, the JDBC URL, and the `DriverManager.getConnection` method call as follows:

```
private String JDBC_DRIVER_CLASS = \
"com.ibm.db2.jcc.DB2Driver";
private String AUCTION_JDBC_URL = \
"jdbc:db2://hostname:50000/auct_svr";
...
connection = DriverManager.getConnection(AUCTION_JDBC\
_URL, AUCTION_DB_USER, AUCTION_DB_PASSWORD);
```

Web service methods

Writing a method that will be invoked by a Web service is similar to writing a method that could be invoked by a regular Java program. The key difference is that arrays are used instead of list containers that contain objects (such as vectors). When using arrays of a particular object type over vectors, the tooling used later on is able to describe the object type contained in a list. Other methods that can be invoked by a Web service are provided in the sample source code. One method, illustrated in the next section, should be sufficient to convey the concepts.

The search method

The implementation of the search method is detailed below. In production implementations where performance is an issue, consider researching indexes and database performance tips. Alternatives for case-insensitive searches using DB2 are detailed in the developerWorks article "Making DB2 Case-Insensitive" (see [Resources](#)).

```
public AuctionListItem[] searchAuctions(String shortNameText) {
    AuctionListItem [] alis;
    alis = new AuctionListItem[0];
    Vector vAlis = new Vector();
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    ResultSetMetaData rsmd = null;
    try {
        conn = getDbConnection();
        if (null == conn) {
            return alis;
        }
        final String SEARCH_ITEMS_SQL = \
```

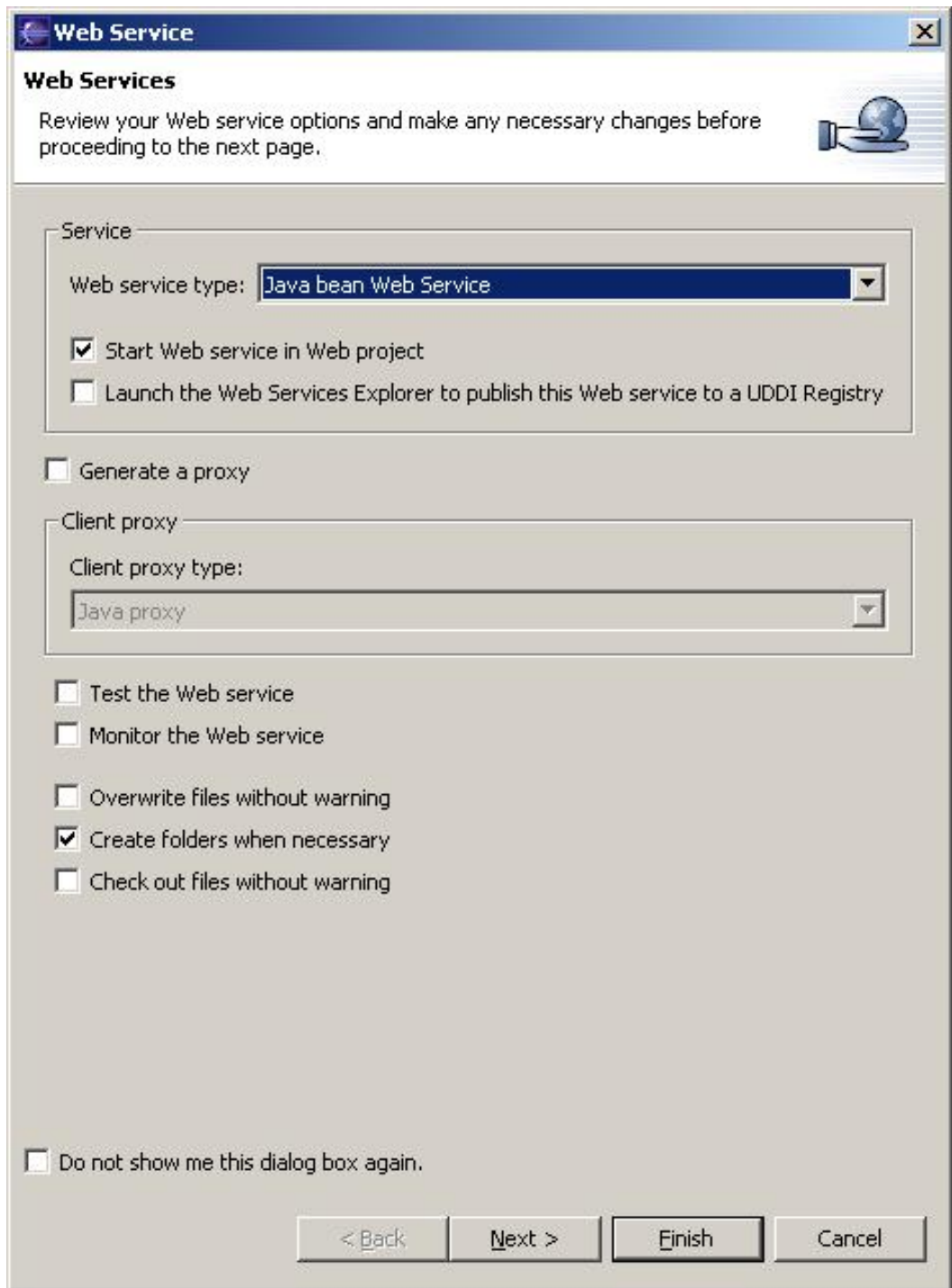
```
"SELECT AUCTION_ID,SHORT_NAME,\
CUR_PRICE,\
END_TIME,STATUS FROM AUCTION.\
AUCTION_ITEM WHERE SHORT_NAME LIKE ?";
pstmt = conn.prepareStatement(SEARCH_ITEMS_SQL);
final String sns = '%' +shortNameText+'%';
pstmt.setString(1,sns);
rs = pstmt.executeQuery();
rsmd = rs.getMetaData();
while(rs.next()) {
    AuctionListItem ali = new AuctionListItem();
    ali.setAuctionId(rs.getInt(1));
    ali.setShortName(rs.getString(2));
    ali.setCurrentPrice(rs.getDouble(3));
    GregorianCalendar gcEnd = new GregorianCalendar();
    gcEnd.setTime(rs.getTimestamp(4));
    ali.setEndTime(gcEnd);
    ali.setStatus(rs.getString(5));
    vAlis.add(ali);
}
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    try { rs.close(); } catch (Exception e) { }
    try { pstmt.close(); } catch (Exception e) { }
    try { conn.close(); } catch (Exception e) { }
}
alis = new AuctionListItem[vAlis.size()];
for(int i = 0; i<vAlis.size(); i++) {
    alis[i] = (AuctionListItem)vAlis.elementAt(i);
}
return alis;
}
```

Section 4. Building the Web service

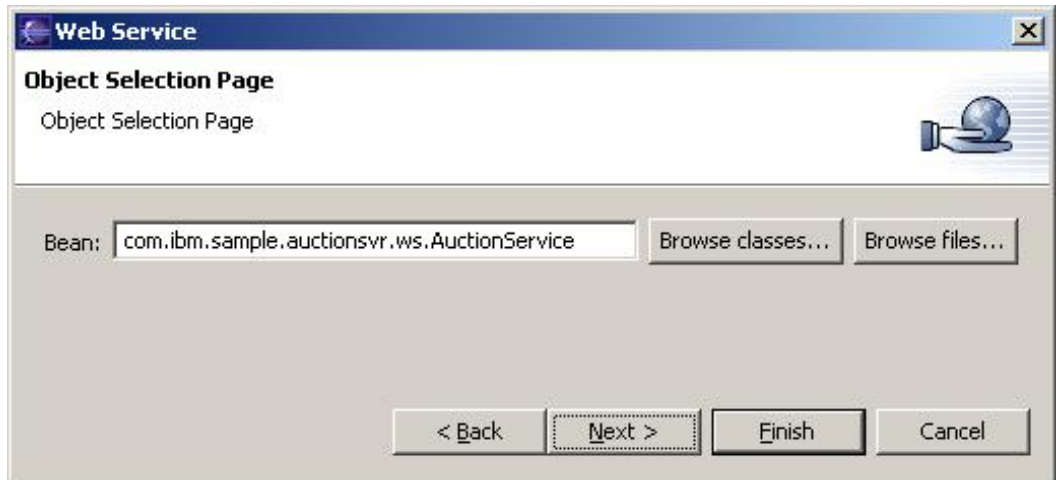
Create a Web service from the Java files

Thus far, we have done nothing specific to the Web service. The steps below will enable your Java code to be invoked by a Web service.

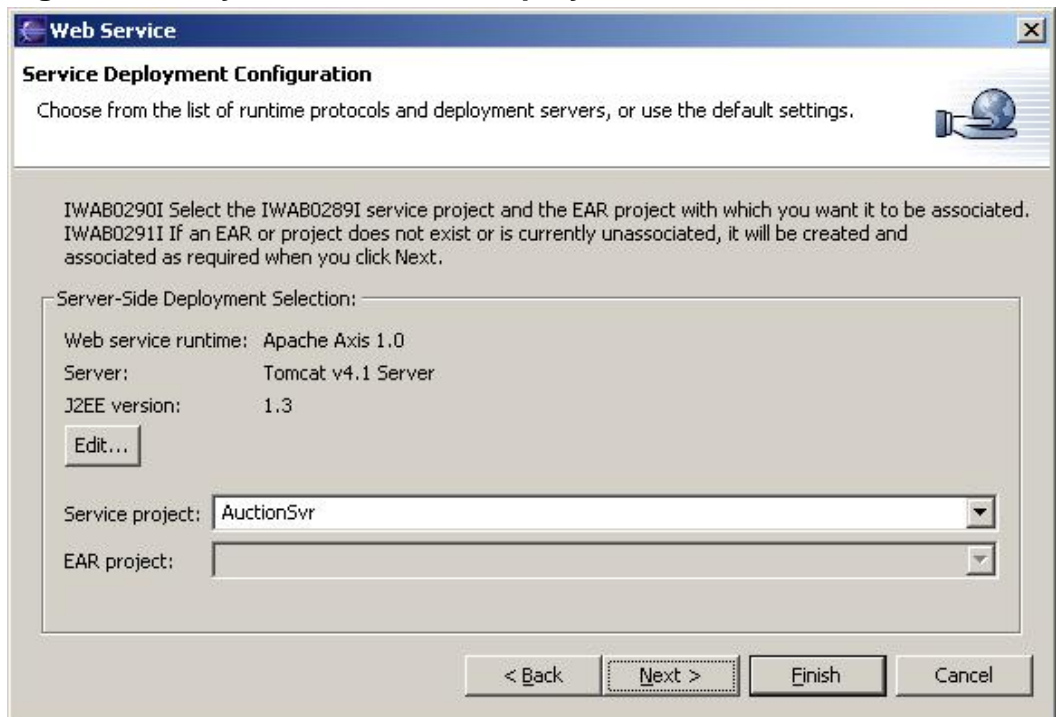
1. To ensure that all the .java files are compiled, select **Project > Clean...**, select **Clean all projects**, then click **OK**.
 2. In the `com.ibm.sample.auctionsvr.ws` package, right-click **AuctionService.java**, and select **Web Services > Create Web Service**.
 3. Keep the default options, then click **Next**.
- Figure 7. Enable Java code to be invoked by a Web service**



4. On the Object Selection Page, keep the default bean of **com.ibm.sample.auctionsvr.ws.AuctionService**, then click **Next**.
Figure 8. Object Selection Page



5. Verify the server-side deployment selection, then click **Next**.
Web service runtime: **Apache Axis 1.0**
Server: **Tomcat v4.1 Server @ localhost**
J2EE version: **1.3**
Service project: **AuctionSvr**
Figure 9. Verify the server-side deployment selection

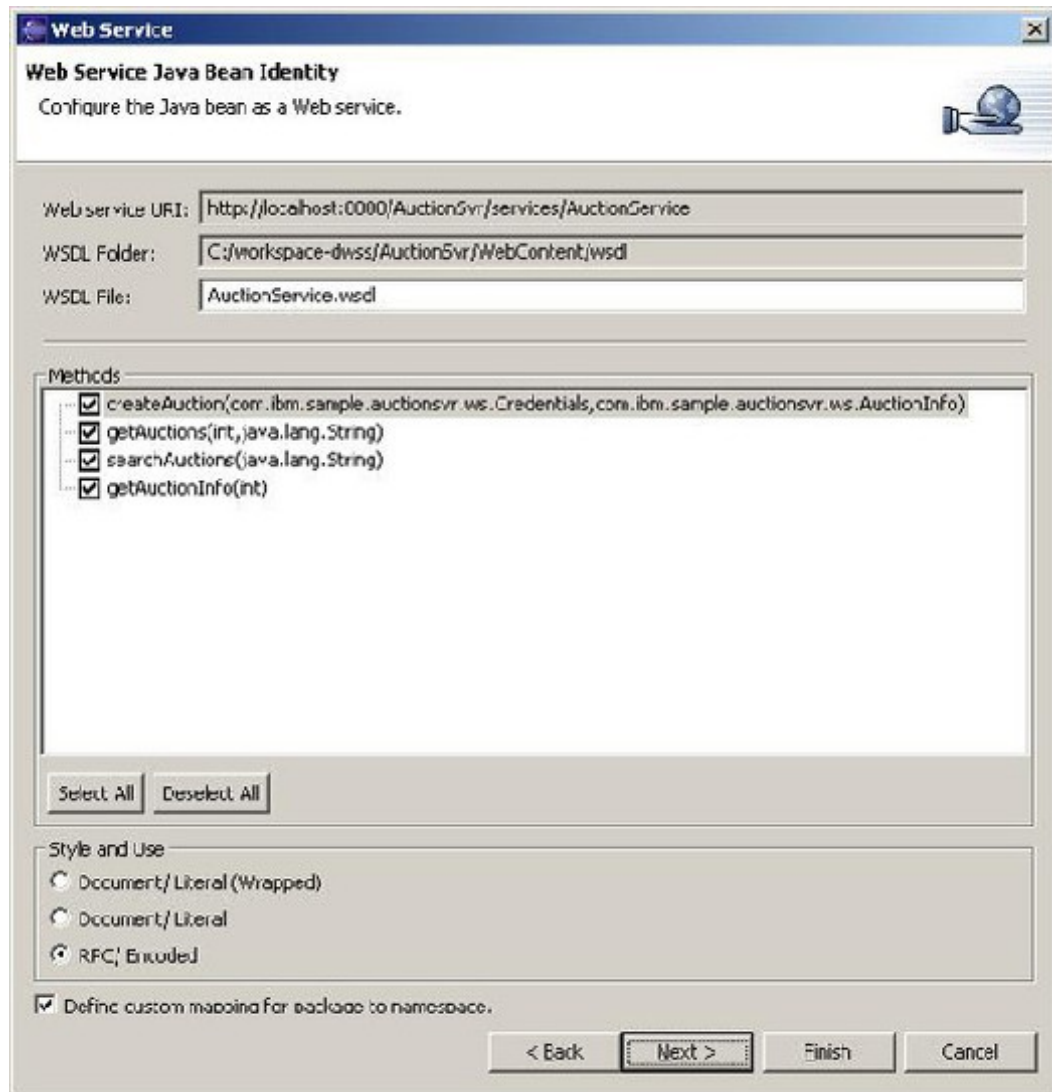


6. Ignore warning messages regarding WS-I Simple SOAP Basic Profile and WS-I-Attachment Profile compliance.
7. Ensure that the **createAuction**, **getAuctions**, **searchAuctions**, and

getAuctionInfo methods are selected.

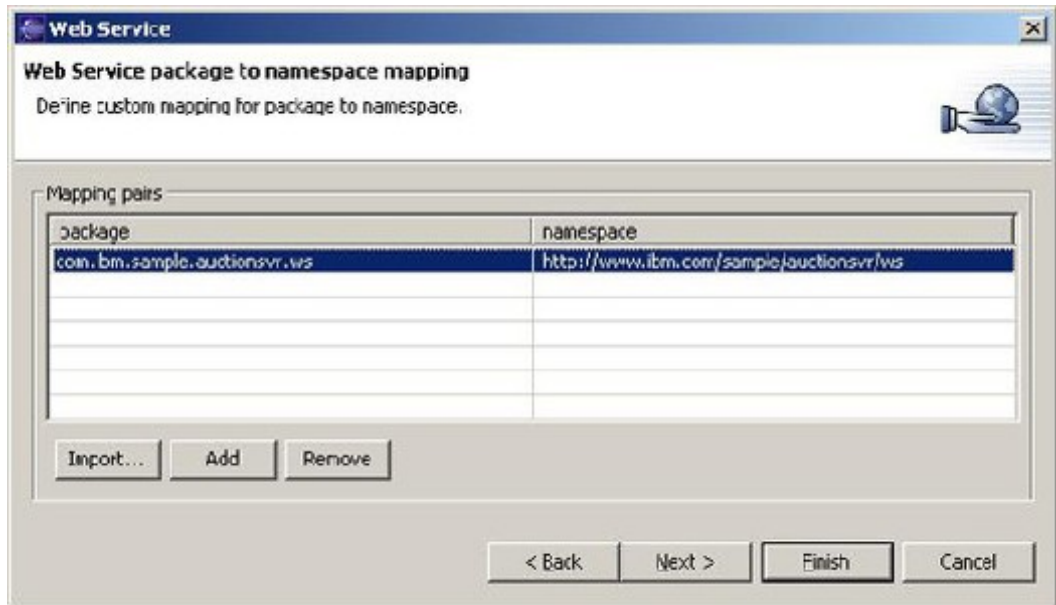
Select **Define custom mapping for package to namespace**, then click **Next**.

Figure 10. Define custom mapping for package to namespace



8. Click **Add**.
9. Specify a package/namespace pair of **com.ibm.sample.auctionsrv.ws** and **http://www.<YourCompany>.com/auctionsrv/ws**.

Figure 11. Specify package/namespace pair

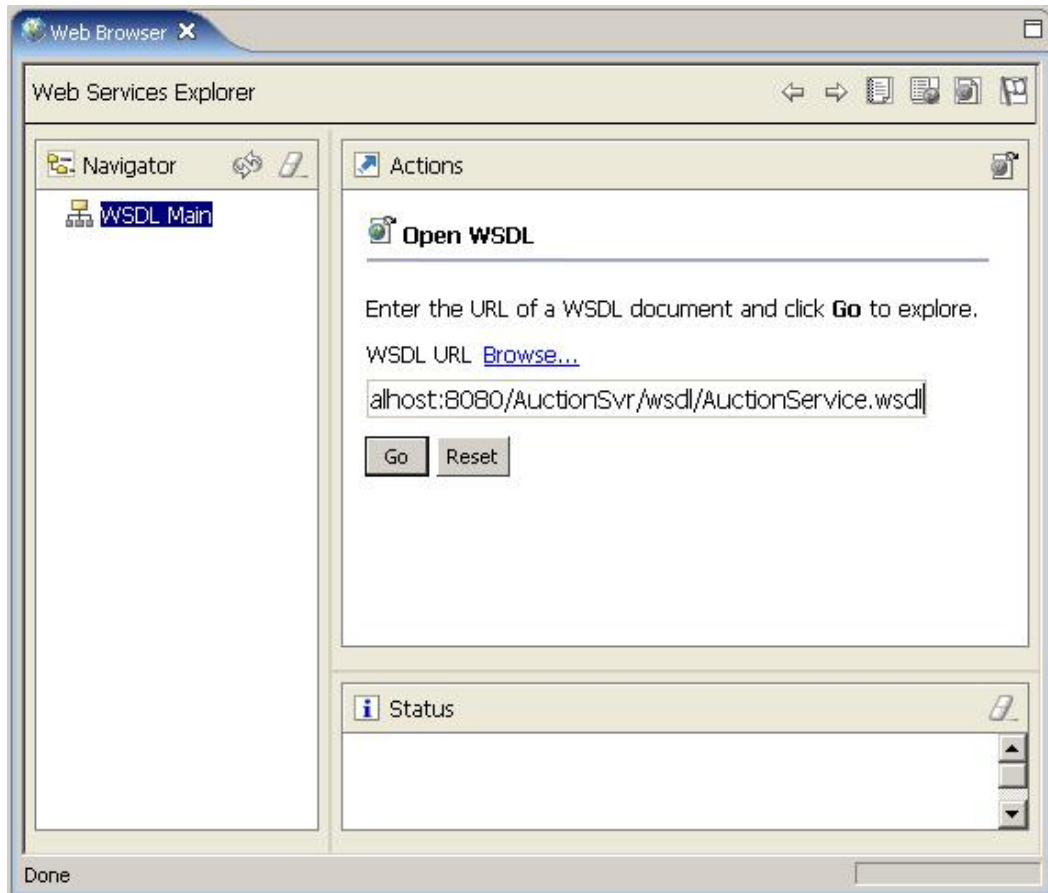


10. Click **Next**, then be patient as files are generated and deployed to Tomcat.
11. On the Web Service Publication wizard page, click **Finish**, leaving both publishing options unchecked.

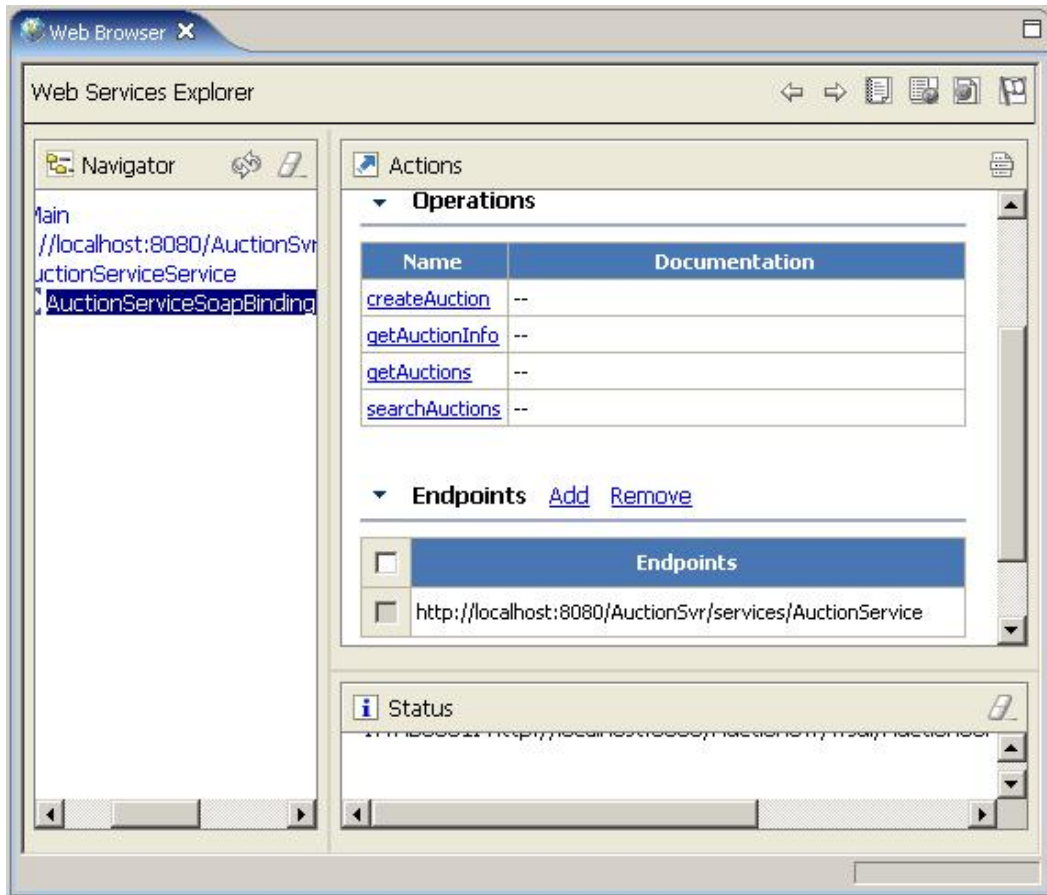
Test the Web service using the Web Services Explorer

1. Select **Run > Launch the Web Services Explorer**.
2. Select the **WSDL Page icon** (second from the right) in the Web Browser/Web Services Explorer.
3. Select the **WSDL Main link** in the Navigator frame.
4. Specify a WSDL URL of **http://localhost:8080/AuctionSvr/wsd/AuctionService.wsdl**, then click **Go**.

Figure 12. Specify WSDL URL



5. Select the **getAuctions** operation.
 6. Specify a **seller ID** of **1**, **Status** of **ALL**, then click **Go**.
- Figure 13. Specify seller ID**



- The auction items that were imported earlier are returned in the Status window.

Figure 14. Status window results



Section 5. Summary

In this tutorial, you created a Web service that interacts with a database.

From here, you can create a Web service client using Eclipse with IBM's Web Tools Platform Contribution or IBM Rational Web Developer for WebSphere Software. See [Resources](#) for another developerWorks tutorial that shows how to build a Cloudscape-enabled Web auction client using the Web Tools Platform.

Downloads

Description	Name	Size	Download method
Source code	os-wtpclientwtp-ws-svr.zip	10.3KB	HTTP

[Information about download methods](#)

Resources

- For more on the Eclipse Web Tools Platform, visit the [Eclipse Web Tools Platform](#).
- Read the [Getting Started guide](#) for IBM's contribution to the Eclipse Web Tools Platform.
- The [Apache Tomcat](#) Web site is the community and download site for Tomcat.
- The developerWorks article "[Integrating Cloudscape and Tomcat](#)" describes how to obtain database connections via JNDI.
- The developerWorks article "[Making DB2 Case-Insensitive](#)" describes multiple ways to perform case-insensitive searches in DB2.
- The developerWorks article on [IBM Cloudscape Version 10.0](#) lists the changes in the latest version and provides links to downloads.
- The companion developerWorks tutorial, "[Build a Web-based client with the Eclipse Web Tools Platform](#)" shows how to create a Web-auction client that accesses a Cloudscape database and the auction Web service you created here.

About the author

Michael Donaghy

Michael Donaghy is a solutions developer with the [IBM Solutions Builder Express Portfolio](#) Development in Raleigh, N.C. He has developed both enterprise and mid-market solutions for IBM. He holds a master's degree in computer science from North Carolina State University. In his spare time, he enjoys waterskiing.