Pace University
DigitalCommons@Pace

CSIS Technical Reports

Ivan G. Seidenberg School of Computer Science and Information Systems

3-1-2003

Java Servlets, Access Databases, and a Stand Alone Server

Carol E. Wolf
Pace University

Follow this and additional works at: http://digitalcommons.pace.edu/csis_tech_reports

Recommended Citation
Wolf, Carol E., "Java Servlets, Access Databases, and a Stand Alone Server" (2003). *CSIS Technical Reports*. Paper 10.
http://digitalcommons.pace.edu/csis_tech_reports/10

This Article is brought to you for free and open access by the Ivan G. Seidenberg School of Computer Science and Information Systems at DigitalCommons@Pace. It has been accepted for inclusion in CSIS Technical Reports by an authorized administrator of DigitalCommons@Pace. For more information, please contact rracelis@pace.edu.

SCHOOL OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

TECHNICAL REPORT

Number 187, March 2003

Java Servlets, Access Databases, and a Stand Alone Server

Carol E. Wolf

PACE UNIVERSITY

Professor Wolf acknowledges with gratitude the contributions to this manuscript from her colleague Narayan Murthy, Professor of Computer Science and Chairperson of the Department of Computer Science on Pace University's Westchester campus. She hails as extraordinarily helpful the instructional materials on servlets authored by Professor Murthy for the Pace Computer Learning Center.

Carol E. Wolf received a BA from Swarthmore College and an MA and Ph.D. from Cornell University, all in mathematics. Her dissertation was on partial recursive functions and recursively enumerable sets. She taught at the New York State University College at Brockport and at Iowa State University before coming to Pace in 1986. From 1988 through 2000 Carol served as chairperson of the Computer Science Department on the New York campus.

Carol's first involvement in computing began while at Swarthmore during a summer internship that had her working on the UNIVAC I, which was one of the world's first commercial digital computers. At Iowa State she become interested in cellular automata and graph grammars. There she cultivated a general expertise in computer science, a field not yet developed when she was in graduate school.

Java Servlets, Access Databases, and a Stand Alone Server

Stand Alone Server

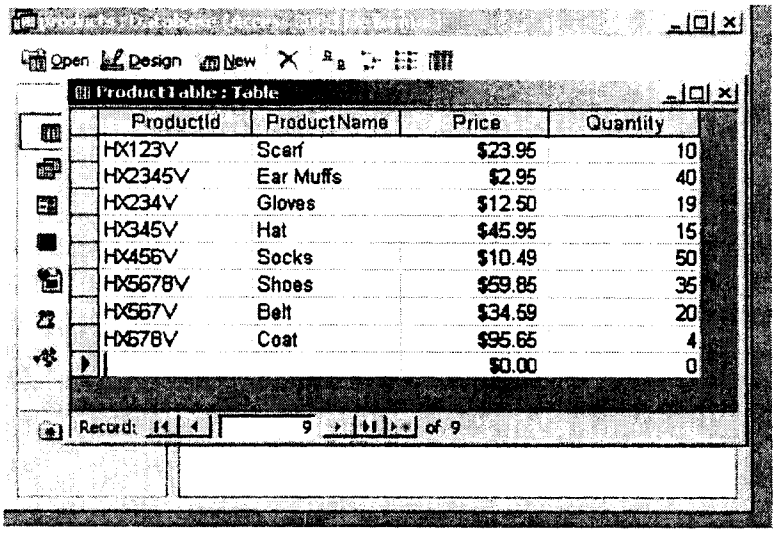
A free stand alone server can be downloaded from the Apache open software project. It is Tomcat 4.0.4, and it is available from http://jakarta.apache.org/ . Directions for installation can be found at http://archive.coreservlets.com/Using-Tomcat.html. Make sure that you download the regular version, not the LE version. The latter requires an XML parser and doesn't work without it.

The directions at coreservlets also tell you which directories to use for your html files and Java servlets. The html files should be placed in the webapps/ROOT subfolder of the Apache folder. The servlets class files belong in the webapps/ROOT/WEB-INF/classes folder. If you are reading from a file, the file goes into that folder also.

Access Databases

Microsoft Access is a widely used personal computer database. In it, you can create linked tables that consist of rows and columns. The columns have field names such as ProductName and Quantity. The rows store the data for each product. Note that there should not be any spaces in the field names.

The *products* database used in this example is very simple. It contains just one table, ProductTable, which is shown below. The field names are ProductId, ProductName, Price, and Quantity. There is very little data in the table. However, it is enough to demonstrate the action of a servlet.



To connect to the database using a Java servlet, you must first register the database with the operating system. In Windows 98 this is done by clicking on **Settings/Control Panel/Data Sources (ODBC)**. In Windows 2000 or XP, you will find this same file in **Settings/Control Panel/Administrative Tools.** Select **Add/Microsoft Access Driver (*.mdb),** and from there **Browse** to find the location of your database.

The connection is done with a jdbc-odbc bridge. Jdbc stands for Java database connectivity API (application programmer interface), while the 'O' in Odbc stands for Open. Odbc is a protocol from Microsoft that is based on the X/Open SQL specification. In a Java servlet, we create a *Connection* object. The lines of code required are:

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection ("jdbc:odbc:products");
```
where *products* is the name used for the database in the registration information.

The Connection and DriverManager objects are contained in the *sql* package, so *import java.sql.*;* must be placed at the beginning of the servlet. SQL stands for *Structured Query Language*. SQL is the standard way to interact with databases. It is not case sensitive, so you can mix upper and lower cases, but commands often use upper case. Some examples are Select, Insert, Delete, and Update. You can also modify commands by using connectors such as Where or Set. An example is

```
Select * From ProductTable Where ProductId = 'orderId'
```
This says to select all (*) the records from the ProductTable that have the given ProductId. Another example is

```
Update ProductTable Set Quantity = 'newQuantity' Where ProductId = 'orderId'
```
This statement updates the Quantity field in the record in ProductTable that has orderId as its ProductId.

HTML Forms

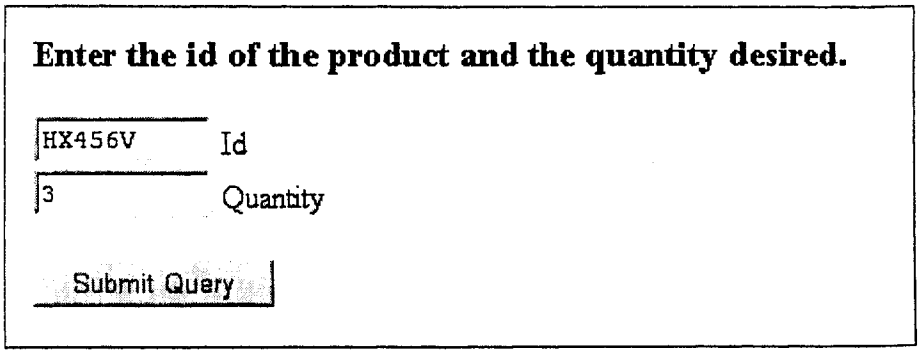
The interface to a servlet is an html (hypertext markup language) form. A simple example is:

```
<html>
<head><title>Order Form</title></head>
<body>
    <h3>Enter the id of the product and the quantity desired.</h3>
    <form method = "get"
        action="http://localhost/servlet/MakeOrderDB/" >
        <input name="id" type="text" value="" size = 10> Id
        <br><input name="quantity" type="text" value="" size = 10> Quantity
        <p><input type="submit">
    </form>
</body></html>
```

The form has a head and a body, both enclosed by appropriate tags. The head for this example only contains the title. The title should be somewhat descriptive of the purpose of the form. In the form, all the work is done in the body. This begins with some information for the client. This tells the client what to do with the form. Here it explains that product information should be entered into the text fields. These are defined between the form tags. The method for this form is "get". If you are using servlets the "post" method would work just as well.

The action value is more complicated. It has to tell the form where the servlet is that will be used to process the form. In this case, the servlet is on the same computer as the html form, so it is accessed using http://localhost/. The remainder of the action string says to look for a servlet called *MakeOrderDB*. The classes for this servlet are stored in the folder described above, and the servlet class itself is called MakeOrderDB. It is accessed from your browser using http://localhost/MakeOrderDB.html.

The remainder of the form is filled by three *input* tags, the first two defining text fields and the third one a *Submit* button. The form is shown below after some data has been entered.



Both input tags contain name fields. These names are used by the servlet to distinguish between the different fields. The URL string that is created when the Submit button is clicked looks like the following:

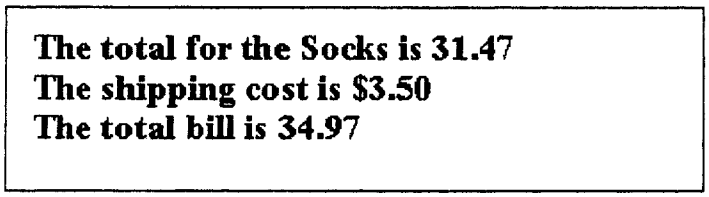
http://localhost/servlet/MakeOrderDB/?id=HX456V&quantity=3

This string begins with the address found in the action string. It then is followed by the values submitted for the id and quantity separated by an '&'. The names of the text fields are listed in order. These become parameters for the servlet.

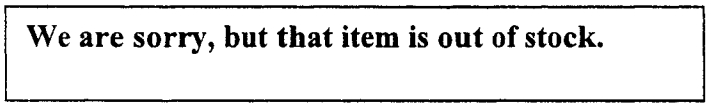
In general, the URL string is coded with all spaces replaced by the '+' sign, and data items separated by ampersands (&). Letters and digits are not changed, but a number of other characters are replaced by the percent sign (%) followed by the ascii code for the character. For example, the 'at' sign (@) is replaced by %40.

Java Servlets

Java servlets are used to process the parameters sent by the form. The servlet that processes the form above first looks in the database to find the product with the given id. If this is found, it then checks to see if the quantity of the item in stock is greater than or equal to the quantity ordered. If so, it uses the price stored with the product to compute the total bill. In this case the servlet will send back the following html page to the client. It also updates the quantity in the database to reflect the sale.



If the quantity is larger than the amount in stock a different message will be displayed.



An actual e-commerce web site would certainly have a much more elaborate response than either of these.

The main class of a servlet extends the HttpServlet class. This class is contained in a file called *servlet.jar* that is available for downloading from SUN. To make things easier for you, it can also be found in the Documents directory of the website found at http://csis.pace.edu/~wolf/. You can download it from there and store it with the other .jar files in the *lib* subdirectory of the jdk folder.

You also have to tell the IDE you use to look there for this file. In JCreator, first click on **Configure**, then on **Options...**, and then on **Profiles**. You should see the version of JDK that you are using listed there. Click on it and then on **Edit**. When you see the next window, click on **Add** and then **Add Package...** Browse until you find the lib subdirectory and then click on servlet.jar. This will tell the compiler where to find the HttpServlet classes needed to compile a servlet.

The HttpServlet classes are a part of javax, extended java. So the import statements for it are

```
import javax.servlet.*;
import javax.servlet.http.*;
```

Like any import statements, these go at the beginning of the file. You need further import statements as well, such as import java.io.* and import java.sql.*.

Servlets look much like applets at the beginning. A servlet class extends HttpServlet where an applet class extends Applet. An example of a class declaration is

```
public class MakeOrderDB extends HttpServlet
```

The principle methods in a servlet class are doGet and doPost. They both have the same parameters, HttpServletRequest and HttpServletResponse. The doGet will usually look like the method below:

```
protected void doGet (HttpServletRequest request, HttpServletResponse response)
{   try
    {
        // Code using request and response.
    }
    catch (NumberFormatException nfexp) {out.println ("Number format exception");}
    catch (IOException ioexp) {out.println ("IO exception");}
} // method doGet
```

Catching the IOException is required. The NumberFormatException would be needed if any of the parameters have to be converted to doubles or integers. All parameters are sent to the servlet as Strings.

Servlets can instantiate other classes and use their public methods. They can read and write to files, query databases, store data in arrays, tables, or lists, etc. But displaying results requires the servlet to create an html file and send it back to the client. This is the response. Typically the servlet gets an instance of the PrintWriter class, often called *out*.

```
PrintWriter out = response.getWriter ();
```

Strings are written to *out* using *println*, just like with System.out.println (). For example

```
out.println ("<h3>The total bill is " + total + "</h3>" );
```

Notice the html tags inside the string, <h3> and </h3>. All tags and other data that are to be displayed by the client's browser are part of the output string. As usual, make sure that the quotation marks are matched properly.

Sample Java Servlet

The following Java servlet can be used with the stand alone server and the html file above to query the database.

/* This is an example of a Java servlet that connects with an Access database called products. It is called by a web page with a form that has input boxes for the product's ID and the quantity to be purchased. The ID is sent to the database, which then returns the record with the given ID. The quantity on hand is then checked against the quantity ordered. If there is enough of the product in the database, the order is processed. The bill is then displayed for the client. If the quantity ordered is too large, an 'Out of Stock' message is sent back to the client. */

```
import java.sql.*;
import java.io.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;

/*  The main servlet class makes the database connection, gets the order from the form and then
makes the order. */
public class MakeOrder extends HttpServlet
{
    private Connection con;
    private PrintWriter out;

    // The database connection is created in the init method.
    public void init ()
    {
        try
        {
            // forName is a static method that 'forces' the use of the jdbc-odbc bridge..
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection ("jdbc:odbc:products");
        }  catch (SQLException ex) {out.println ("<h3>SQL Exception.</h3>");}
        catch (ClassNotFoundException cex)
        {out.println ("<h3>Class Not Found Exception.</h3>");}
    } // init


    // The doGet method is used to get the order from the client and process the order.
    protected void doGet (HttpServletRequest request, HttpServletResponse response)
    {
        try
        {
            out = response.getWriter ();
            Order order = new Order (con, out);
            order.getDataFromClient (request);
            order.makeOrder ();
            con.close ();
        } catch (IOException ex) {out.println ("<h3>IO Exception.</h3>");}
        catch (ServletException se) {out.println ("<h3>Servlet Exception.</h3>");}
        catch (Exception e){out.println ("<h3>Database connection exception.</h3>");}
        {out.println ("<h3>Database connection exception.</h3>");}
    } // doGet

    /* The servlet will send an html response page to the client.  This method creates the header for that
    page. */
    private void createHeader (String title)
    {
        out.println ("<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>");
        out.println ("<HTML>");
```

```
        out.println ("<head>");
        out.println ("<title>" + title + "</title>");
        out.println ("</head>");
        out.println ("<body>");
    } // createHeader

    // The end of the html page consists only of closing tags.
    private void createFoot () {out.println ("</body></html>");      }

} // MakeOrder

/*   The Product class is used to store and process all information about a single product. It has
instance fields for all the fields in a database record. There are methods to get and display a product. */
class Product
{
    private String productName, productId;
    private double price;
    private int quantity;

    protected String getProductName () {return productName;}
    protected double getPrice () {return price;}
    protected int getQuantity () {return quantity;}

    /* The Order class sends a query to the database. The database then responds with a ResultSet.
    The method, getProduct, extracts the individual fields from one record. */
    public void getProduct (ResultSet rs, PrintWriter out)
    {
        try
        {
            productId = rs.getString ("ProductId");
            productName = rs.getString ("ProductName");
            price = rs.getDouble ("Price");
            quantity = rs.getInt ("Quantity");
        }   catch (Exception ex){out.println ("<h3>Connection Error.</h3>");}
    } // method getProduct

    // This method creates html code that can send product information back to the client.
    public void displayProduct (PrintWriter out)
    {
        out.println ("<h4>Product Name: " + productName );
        out.println ("<br>Product Id: " + productId);
        out.println ("<br>Product Price: " + price );
        out.println ("<br>Product Quantity: " + quantity + "</h4>");
    } // method displayProduct
}// class Product

/* The Order class processes the data sent in by the client. It gets the data and uses it to create a query.
The query is then submitted to the database. The ResultSet is then used to get and display the product.
If there is sufficient quantity, the order is processed and an order form is created. This is then returned
to the client. /*
```

```
class Order
{
    private Connection con;
    private PrintWriter out;
    private Product product;
    private String orderId;
    private int quantityOrdered, quantityInStock;

    Order (Connection con, PrintWriter out)
    {
        this.con= con;
        this.out = out;
    } // constructor

    // The servlet request is used to get the data from the form submitted by the client.
    public void getDataFromClient (HttpServletRequest request)
    {
        try
        {
            orderId = request.getParameter ("productId");
            quantityOrdered = Integer.parseInt (request.getParameter ("quantity"));
        } catch (NumberFormatException e){out.println ("Number error.");}
    } // getDataFromClient

    /* A query is sent to the database to return a record with the database ProductId equal to the
    orderId submitted by the client. If successful, a new product is created in which to store the data
    from the query. The product is displayed to aid the program developer. */
    public void makeOrder () throws Exception
    {
        Statement stmt = con.createStatement ();
        String query = "Select * From ProductTable Where ProductId = '" + orderId + "'";
        ResultSet rs = stmt.executeQuery (query);
        if (rs.next ())
        {
            product = new Product ();
            product.getProduct (rs, out);
            quantityInStock = product.getQuantity ();
            if (quantityInStock >= quantityOrdered)
            {
                product.displayProduct (out);
                createOrderForm ();
                updateDataBase ();
            }
            else out.println ("<h3>Out of Stock.</h3>");
        }
        else out.println ("<h3>Invalid product Id.</h3>");
    } /
```

```
/* After an order has been made, the quantity of the product in the database has to be adjusted to
take into account that some of it has been sold. This method sends a query to the database telling it
to insert a new quantity into the quantity field of the product. */
 private void updateDataBase () throws Exception
 {
      int newQuantity = quantityInStock - quantityOrdered;
      Statement stmt = con.createStatement ();
      String query = "UPDATE ProductTable SET Quantity = '" + newQuantity
                  + "' WHERE ProductId = '" + orderId + "'";
      stmt.executeUpdate (query);
 } // updateDataBase

 /* The order form consists of the product information followed by the bill. The bill consists of the
price times the quantity ordered plus the shipping cost. Shipping is set at $3.50 for this example. */
 private void createOrderForm ()
 {
      out.println ("<h3>Your Order Id: " + orderId);
      out.println ("<br>The Quantity Ordered: " + quantityOrdered);
      out.println ("<br>The Product Ordered: " + product.getProductName ());
      out.println ("<br>The Product's Price: " + product.getPrice () + "</h3>");
      if (quantityOrdered > product.getQuantity ())
           out.println ("<h3>Sorry, out of stock.</h3>");
      else
      {
           double bill = product.getPrice () * quantityOrdered;
           double total = bill + 3.50; // The shipping cost is set at $3.50.
           out.println ("<h3>The total for the " +
                product.getProductName () + " is " + decimals (bill));
           out.println ("<br>The shipping cost is $3.50");
           out.println ("<br>The total bill is " + decimals (total) + "</h3>" );
      }
 } // createOrderForm

 // The method, decimals, is used to format the output with two decimal places.
 private String decimals (double num)
 {
      DecimalFormat decFor = new DecimalFormat ();
      decFor.setMaximumFractionDigits (2);
      decFor.setMinimumFractionDigits (2);
      return decFor.format (num);
 } // decimals
} // class Order
```

References

1. Deitel, H.M., P.J. Deitel, and T.R. Nieto, *Internet and World Wide Web: How to Program*, Prentice Hall, 2nd Edition, 2002.

2. Horstmann, Cay S. and Gary Cornell, *Core Java Volume II – Advanced Features*, The Sunsoft Press, Java Series, 1998.

3. Murthy, Narayan, *Java Servlets,* Pace Computer Learning Center, 2000..

4. Wigglesworth, Joe and Paula Lumby, *Java Programming: Advanced Topics*, Course Technology, 2000.

School of Computer Science and Information Systems
Pace University
Technical Report Series

EDITORIAL BOARD

Editor:
Allen Stix, Computer Science, Pace--Westchester

Associate Editors:
Connie Knapp, Information Systems, Pace--New York
Susan M. Merritt, Dean, SCSIS--Pace

Members:
Howard S. Blum, Computer Science, Pace--New York
Donald M. Booker, Information Systems, Pace--New York
M. Judith Caouette, Office Information Systems, Pace--Westchester
Nicholas J. DeLillo, Mathematics and Computer Science, Manhattan College
Fred Grossman, Information Systems, Pace--New York
Fran Goertzel Gustavson, Information Systems, Pace--Westchester
Joseph F. Malerba, Computer Science, Pace--Westchester
John S. Mallozzi, Computer Information Sciences, Iona College
John C. Molluzzo, Information Systems, Pace--New York
Narayan S. Murthy, Computer Science, Pace--New York
Catherine Ricardo, Computer Information Sciences, Iona College
Sylvester Tuohy, Computer Science, Pace--Westchester
C. T. Zahn, Computer Science, Pace--Westchester

The School of Computer Science and Information Systems, through the Technical Report Series, provides members of the community an opportunity to disseminate the results of their research by publishing monographs, working papers, and tutorials. *Technical Reports* is a place where scholarly striving is respected.

All preprints and recent reprints are requested and accepted. New manuscripts are read by two members of the editorial board; the editor decides upon publication. Authors, please note that production is Xerographic from the pages you have submitted. Statements of policy and mission may be found in issues #29 (April 1990) and #34 (September 1990).

Please direct submissions as well as requests for single copies to:

Allen Stix
School of CS & IS - Suite 412 Graduate Center
Pace University
1 Martine Avenue
White Plains, NY 10606-1932