

Teaching Tip

Form Responders: Enhancing Student Learning in Beginning and Advanced Web Development Classes

William L. Lomerson
Information Technology & Operations Management
Walker College of Business
Appalachian State University
Boone, NC 28608
lomersonwl@appstate.edu

ABSTRACT

Throughout the higher education community there is an increasing presence of courses that strive to introduce students to the technology of the e-commerce environment at a variety of instructional levels. The content of these courses ranges from augmenting an introductory course on computer productivity tools with a segment on auto web formatted documents to senior and graduate level courses that address the full three-tier model of an e-commerce system. This paper examines the benefits and shortcomings of the common methods used to demonstrate web site interactivity using HTML forms. The paper also presents a simple and effective instructional tool that enhances the student's understanding and development of interactive web sites using HTML forms. The tool consists of a simple server side ASP program, which can be easily implemented by the instructor, that responds directly and immediately to a form action request submitted by a web user.

Keywords: web teaching tool, Internet development, HTML forms, e-commerce teaching

1. INTRODUCTION

With the proliferation of e-commerce and the utilization of Internet technologies within public and private organizations, higher education faces a growing need to prepare students to work with these evolving technologies. Colleges and universities have responded by offering a variety of courses at various levels throughout the curriculum. The content of these courses ranges from augmenting an introductory course on computer productivity tools with an automatically generated web formatted document to senior and graduate level courses that address the full three-tier model of an e-commerce system. At each of these levels instructors attempt to develop a robust learning environment that includes enabling students to experience web site interactivity.

This paper examines the most common methods of teaching HTML form functionality, which is used extensively in e-commerce applications to provide web-

site interactivity. It also identifies the shortcomings of the common methods for demonstrating this interactive functionality. Following this analysis, a more effective operational solution to demonstrate this functionality is presented. This discussion assumes that the reader has a working knowledge of Hypertext Markup Language (HTML) in general and the FORM tag specifically.

For beginning HTML students, the tool outlined enables them to immediately see the results of the data acquisition and submission features of a correctly built form, just as it would be implemented in a real interactive web environment. For advanced web students, the tool enables them to determine that their form is submitting data as intended in the designed name-value pairs without having their server side processing (database, etc.) completely operational. Determining the data values provided by a form can be a very difficult problem to isolate during the development of advanced client server interactions.

2. BACKGROUND

One of the key benefits of the web environment is the ability to support interactive content. Because of this benefit, any course dealing with the development of a web site quickly advances to the stage of creating an interactive web environment. The first stage of web site interaction occurs when the web client sends a request for a new web page or a different URL to the web server. This action is very simple and achieved using the hyperlink anchor tag, `<a>`.

The second stage of web site interaction occurs when the web user needs to send specific information to a program on the web server. The HTML form tag, `<form></form>`, is the primary element that handles the interactive transfer of user information from the client to the server. While most HTML tags can be adequately demonstrated using only the browser on the client computer, the FORM tag requires an interaction with the Internet environment in order to fully demonstrate its functionality.

The FORM tag has two distinct operating phases: the presentation/data collection phase and the submission/processing phase. The first phase is handled entirely by the browser and its results are displayed in the browser window. The second phase, which is the reason for including forms in the web page, provides for collecting information from the user and submitting it to a non-HTML application for processing. These applications are normally stored on a server that is external to the client (e.g., an Active Server Page (ASP) database maintenance program).

3. COMMON METHODS

Introductory HTML courses and textbooks handle this second phase processing in a variety of ways. Some skirt this issue by stipulating that the processing is being handled by another organizational department and thus can be ignored by the developer/student. This approach is not very satisfactory for most instructional approaches since it does not provide the student with a complete understanding or appreciation of a form's purpose within a web site.

A more widely used approach is to use the "mailto" feature in the action attribute of the form. When the user submits a form with a mailto action, an email message is prepared and sent from the user to the designated email recipient. While this approach has some limited applicability in the "real world," and can therefore be marginally justified pedagogically, it has several significant operational deficiencies.

The most significant deficiency arises from an operational requirement of the "mailto" action. Specifically, the web client machine must have a default email program that has been pre-configured for the specific user submitting the form. A large number, if not most, of the computers accessible to the general student population in open or dedicated labs have operating policies that purposely discourage personalized local email programs. The second deficiency of this approach arises from the time delay, often substantial, that occurs between the submission of the form and the availability of the delivered message for retrieval. The third deficiency results from the FORM tag requirement that the user provide an additional attribute, `enctype="plain/text"`, in order to create a message with readable content. A simple example of a form that uses the mailto action is shown in Figures 1, 2 and 3.

4. ADVANCED METHODS

A better approach for demonstrating form interactivity is to create a server side program that will actually process the form submission and respond directly to the submitter. Some textbooks suggest that the instructor write or acquire such a program or have someone on their technical support staff write one (e.g., Groves, et al. 2000, p. 127-8; Morrison 2001, p. 102-3; Neou, 1999, p. 252-5). Unfortunately, many of the instructors teaching introductory client side scripting do not have the necessary knowledge to write or implement a program of this type and/or the technical staff is often unavailable to develop and support such a program. Figure 4 provides an example of a CGI/Perl script for such a program. However, while this program can run on both Unix and Windows based computers, it usually must be installed and maintained on the web server by the technical support staff.

5. ALTERNATIVE METHOD

Perhaps the best approach for demonstrating web site interactivity is the alternative offered by this paper. This approach uses a simple server side ASP program that responds directly and immediately to the form action request submitted by a web user/student. While this program may not be quite as universal as a CGI/Perl script, it will run on any Windows NT4 or 2000 server running the IIS web server and ASP 2.0 or higher. This configuration is widely used and available on most campuses. Further, this form responder script can be installed and maintained by the instructor using normal access to their web space. The single technical support input required for this program to work is that the web directory, where this ASP program is located, must be enabled to permit script execution. This script is then

accessed by assigning the URL for the form responder script to the form action attribute. (Make sure the form responder script file is saved with an “asp” extension.) Figure 5 presents the basic processing script along with code that displays the results in a readable format.

This program reads and displays the names of the form fields exactly as they are encoded in the form. It then displays the associated value of each field that is submitted by the form for processing. There are no restrictions on the input names or input types (except that “file” input types may not be used) or on the number of inputs that may be submitted to the program. An example of the Form Responder output for the earlier example in Figure 1 is shown in Figure 6. This output was produced by changing the form action in that earlier script to point to the URL containing the FormResponder program (e.g., `<form method="post" action="http://mywebserver.edu/FormResponder.asp">`).

This response to this action is immediate and, in fact, mimics the actual processing of a web server application. The student receives timely feedback to use in correcting any errors as well as gaining the experience of developing a workable interactive web page.

This program also proves useful in advanced web development classes as an aid to debugging sever side programs. In many of these classes students investigate more complex form techniques. These techniques include learning to manipulate the name of the form input tags dynamically and using hidden fields to submit the data for server side processing. Since the results of these manipulations may never present themselves explicitly, it can be very difficult for the student to determine the source of associated processing errors. With this program, the actual names generated by the client side processing, as well as the value associated with the name, are echoed back to the client exactly as they have been created. Actual values can then be compared to the expected values and necessary modifications made in the code. This enables the student to get the client side script working as intended. Once the student is assured that the client side variables are being transmitted as expected, the associated server side script can be developed and debugged with much more confidence and success.

A simple example of a more advanced form created by an ASP script is shown in Figure 7. The data submitted by this form cannot be determined by an examination of the native script. In most cases the information on a form like this is submitted to another ASP script for additional processing. It is generally not echoed back to the calling script. The FormResponder provides a very simple and convenient method by which the

programmer can determine exactly what data is being sent to the processing script. The browser screen generated by this script is shown in Figure 8. Finally, the browser screen generated by submitting this form to the FormResponder is shown in Figure 9.

6. CONCLUSION

This paper has reviewed several common methods for teaching form processing in both beginning and advanced web design classes. It has presented the benefits, shortcomings and examples of each method. Finally, it has proposed a simple method that is easily created by the instructor as well as easily utilized by students.

Actual classroom experience with this program has shown it to be easily incorporated into the most elementary level of HTML coding activities. Students gain immediate appreciation of the instant interactivity that can be provided by a web site. For advanced students, the time savings gained in the debugging process of complex client-server interactions are greatly appreciated.

7. REFERENCES

- Groves, D., J. Finnegan and J. Griffin [2000], *The Web Page Workbook*, 2ed. Franklin, Beedle & Associates, Inc., Wilsonville, OR.
- Negrino, Tom and Dori Smith [2001], *JavaScript for the World Wide Web*. Peachpit Press, Berkeley, CA.
- Neou, Vivian [1999], *HTML 4.0 CD with JavaScript*. Prentice Hall PTR, Upper Saddle River, NJ.

8. RECOMMENDED READING

- Mitchell, Scott and James Atkinson [2000], *SAMS Teach Yourself Active Server Pages 3.0 in 21 Days*. SAMS, Indianapolis, IN.
- Morrison, Michael [2001], *HTML & XML for beginners*. Microsoft Press, Redmond, WA

AUTHOR BIOGRAPHY

William L. Lomerson is an Assistant Professor of Information Technology at Appalachian State University. He holds a Ph.D. from the University of North Texas and an MBA from the University of Texas. He currently teaches several different e-commerce/ web development courses. His research interests are directed at performance assessment of



IT systems and e-commerce/web development pedagogical issues. Dr. Lomerson is Past President and current Executive Board member of the Southern Association of Information Systems, a regional affiliate of AIS. Prior to his academic career, Dr. Lomerson held a variety of systems development positions in industry.

Figure 1 – Simple HTML form script with mailto action

```
<html>
<head>
<title>Email Form Processing</title>
</head>
<body>
<h2>Email Form Processing</h2>
<form method="post" action="mailto:jjones41@myschool.edu?subject=Test email
form" enctype="text/plain">
Full Name: <input type="text" name="FullName"><br><br>
Favorite Animal:<br>
  <input type="checkbox" name="Animal" value="Dog"> Dog<br><br>
  <input type="checkbox" name="Animal" value="Cat"> Cat<br><br>
  <input type="checkbox" name="Animal" value="Gerbil"> Gerbil<br><br>
  <input type="submit">
</form>
</body>
</html>
```

Figure 2 – Browser presentation of simple HTML form script

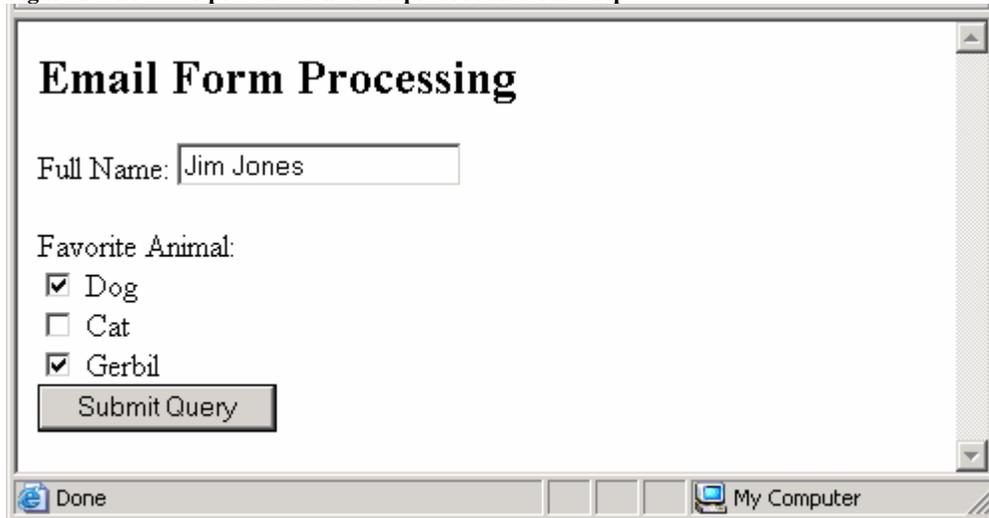


Figure 3 – Email response to form submitted with a mailto form action

```
From: "Jim Jones" <jjones41@myschool.edu>
To: <jjones41@myschool.edu>
Sent: Thursday, November 15, 2001 1:39 PM
Subject: Test email form
```

```
FullName=John Jones
Animal=Dog
Animal=Gerbil
```

Figure 4 – CGI/Perl script for a form responder

```
print "Content-type: text/html\n\n";
print <HTML><HEAD><TITLE>Sample script response</TITLE>
</HEAD><BODY\n\n";
print "<H1>Response from sample form</h1\n\n";

if ($E'V{'REQUEST_METHOD'} "q" "POST") {
    $form = <STDIN>;          # get form data
    $form =~ s/\s//g;        # remove white space
    $form =~ s/%([0-9a-f]{1,2})/pack(C,hex($1))/eig;
    # convert escaped characters
    $form =~ s/\+/ /g;      # convert '+' characters to spaces

    @entries = split(/&/, $form);
    # form fields are separated 'y' '&' characters
    #create an array of the field/value pairs

    foreach $entry (@entries) {
        # print out the field/value pairs
        ($item, $value = split(/=/, $entry);
        print "<BR>The value of <B>$item<?B> is <B>$value</B\n\n";
    }
}
else {
    print "<P>sorry, this script only supports the
    the POST method\n\n";
}
print "</BODY><HTML\n\n";
```

Source: Groves, D., Finnegan, J., and Griffin, J. *The Web Page Workbook, 2ed.*, p. 127-8.

Figure 5 – ASP code for form responder program

```
<html>
<head>
<title>Form Responder</title>
</head>
<body>
<table align="center">
<tr><td colspan="3"><h3>Echo of Submitted Form Information</h3></td></tr>
<%
'--The actual form responder code
count = Request.Form.Count
for k = 1 to count
    Response.write("<tr><td align='right'>" & Request.Form.Key(k) & ":</td>")
    Response.write("<td width='5'></td><td>" & Request.Form(k) &
"</td></tr>")
next
%>
</table>
</body>
</html>
```

Figure 6 – Browser output from FormResponder.asp action

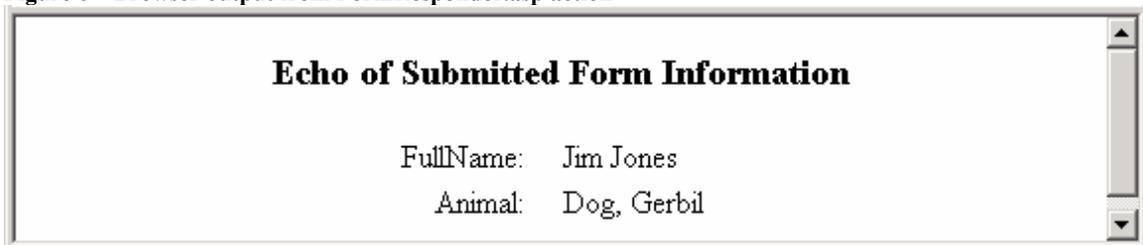


Figure 7 - ASP Script for an advanced form

```
<html><head>
  <title>Advanced Form</title>
</head>
<body>
  <h2>Request for Additional Information</h2>
  (CTRL-click to select multiple categories)<br><br>
  <%
    set oRS = Server.createobject("ADODB.recordset")
    oRS.open "SELECT * FROM Categories", "DSN=Nwind" %>
  <form action="http://mywebserver.edu/formresponder.asp" method="post">
    <input type="hidden" name="RequestorIP" value="<%=
request.servervariables("REMOTE_ADDR") %>">
    <input type="hidden" name="Requested" value="<%= now %>">
    <select name="Categories" multiple size="8">
  <%
    while (not oRS.EOF)%>
    <option value="<%= oRS("CategoryID") %>"><%= oRS("CategoryName")
%></option><br>
  <%
    oRS.movenext
    wend %>
  </select>
  <input type="submit" value="Submit Request">
</form>
<% oRS.close %>
</body></html>
```

Figure 8 – Browser presentation of advanced ASP form script

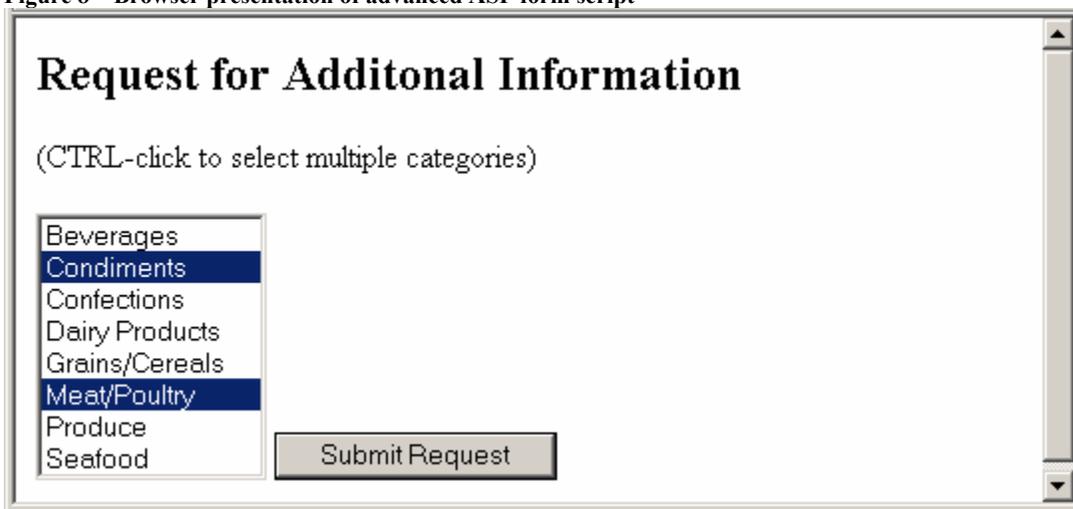
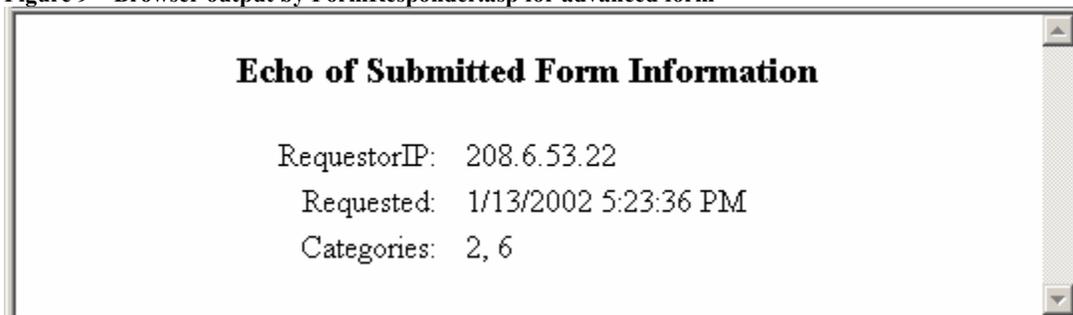


Figure 9 – Browser output by FormResponder.asp for advanced form





STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

Copyright ©2002 by the Information Systems & Computing Academic Professionals, Inc. (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to the Editor-in-Chief, Journal of Information Systems Education, editor@jise.org.

ISSN 1055-3096