FORM VALIDATION

Himanshu Kapoor
Dronacharya College Of Engineering, Khentawas, Gurgaon

I. INTRODUCTION

The forms in HTML provide a very simple and reliable user interface to collect data from the user and transmit the data to a servlet or other server-side program for processing. In this chapter we present the standard form controls defined by the HTML 4.0 specification. However, before covering each control, we first explain how the form data is transmitted to the server when a GET or POST request is made. We also present a mini Web server that is useful for understanding and debugging the data sent by your HTML forms. The server simply reads all the HTTP data sent to it by the browser, then returns a Web page with those lines embedded within a PRE element. We use this server throughout the examples in this chapter to show the form control data that is sent to the server when the HTML form is submitted. To use forms, you'll need to remember where to place regular HTML files to make them accessible to the Web server. Below, we review the location for HTML files in the default Web application for Tomcat, JRun, and Resin.

1.1 How HTML Forms Transmit Data

Each of the HTML controls typically has a name and a value, where the name is
specified in the HTML and the value comes either from user input or from a default
value in the HTML.HTML forms let you create a variety of user interface controls to collect input in a Web page. The entire form is associated with the URL of a program that
will process the data, and when the user submits the form (usually by pressing a button),
the names and values of the controls are sent to the designated URL as a string
of the form.
name1=value1&name2=value2...&nameN=valueN
This string can be sent to the designated program in one of two ways: GET or
POST. The first method, an HTTP GET request, appends the form data to the end of
the specified URL after a question mark. The second method, HTTP POST, sends
the data after the HTTP request headers and a blank line. In the following examples,
we show explicitly how the data is sent to the server for both GET and POST requests.

For example, Listing 1.1 (HTML code) and Figure 1.1 (typical result) show a
simple form with two textfields. The HTML elements that make up this form are discussed
in detail in the rest of this chapter, but for now note a couple of things. First,
observe that one textfield has a name of firstName and the other has a name of
lastName. Second, note that the GUI controls are considered text-level (inline) elements,
so you need to use explicit HTML formatting to make sure that the controls
appear next to the text describing them. Finally, notice that the FORM element designates
http://localhost:8088/SomeProgram as the URL to which the data will be sent.
Before submitting the form, we started a server program called EchoServer on
port 8088 of our local machine. EchoServer, shown in Section 19.12, is a mini Web
server used for debugging. No matter what URL is specified and what data is sent to
EchoServer, it merely returns a Web page showing all the HTTP information sent
by the browser.

Listing 1 GetForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
```

```
<HEAD>
<TITLE>A Sample Form Using GET</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H2>A Sample Form Using GET</H2>
<FORM
ACTION="http://localhost:8088/SomeProgram">
First name:
<INPUT TYPE="TEXT" NAME="firstName"
VALUE="Joe"><BR>
Last name:
<INPUT TYPE="TEXT" NAME="lastName"
VALUE="Hacker"><P>
<INPUT TYPE="SUBMIT"> <!-- Press this button
to submit form -->
</FORM>
</CENTER>
</BODY></HTML>
```

Listing 1.2 (HTML code)

Show a variation that uses POST instead of GET. As shown in Figure 19–4, submitting the form with textfield values of Joe and Hacker results in the line firstName=Joe&lastName=Hacker
being sent to the browser on a separate line after the HTTP request headers and a
blank line.

That's the general idea behind HTML forms: GUI controls gather data from the
user, each control has a name and a value, and a string containing all the name/value
pairs is sent to the server when the form is submitted. Extracting the names and values
on the server is straightforward in servlets. The commonly used form controls are
covered in the following sections.

Listing 1.2 PostForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>A Sample Form Using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H2>A Sample Form Using POST</H2>
<FORM
ACTION="http://localhost:8088/SomeProgram"
METHOD="POST">
First name:
<INPUT TYPE="TEXT" NAME="firstName"
VALUE="Joe"><BR>
Last name:
<INPUT TYPE="TEXT" NAME="lastName"
VALUE="Hacker"><P>
<INPUT TYPE="SUBMIT">
</FORM>
</CENTER>
</BODY></HTML>
```

1.2 The FORM Element

Each of the elements is typically given a name in the HTML source code,
and each has a value based on the original HTML or user input.HTML forms allow you to create a set of data input elements associated with a particular URL.. When the form is
submitted, the names and values of all active elements are collected into a string with
= between each name and value and with & between each name/value pair. This
string is then transmitted to the URL designated by the FORM element. The string is
either appended to the URL after a question mark or sent on a separate line after the
HTTP request headers and a blank line, depending on whether GET or POST is used
as the submission method. This section covers the FORM element itself, used primarily
to designate the URL and to choose the submission method. The following sections
cover the various user interface controls that can be used within forms.

HTML Element: <FORM ACTION="..." ...> ... </FORM>
Attributes: ACTION, METHOD, ENCTYPE, TARGET, ONSUBMIT, ONRESET,
ACCEPT, ACCEPT-CHARSET
The FORM element creates an area for data input elements and designates the URL
to which any collected data will be transmitted. For example:

<FORM
ACTION="http://some.isp.com/someWebApp/SomeServlet">
FORM input elements and regular HTML
</FORM>
The rest of this section explains the attributes that apply to the FORM element:
ACTION, METHOD, ENCTYPE, TARGET, ONSUBMIT, ONRESET, ACCEPT, and
ACCEPT-CHARSET. Note that we do not discuss attributes like STYLE, CLASS, and
LANG that apply to general HTML elements, but only those that are specific to the
FORM element.

II. ACTION

The ACTION attribute specifies the URL of the server-side program that will
process the FORM data (e.g., http://www.whitehouse.gov/servlet/schedulefund-raiser). If the server-side program is located on the same server from
which the HTML form was obtained, we recommend using a relative URL
instead of an absolute URL for the action. This approach lets you move both
the form and the servlet to a different host without editing either. This is an
important consideration since you typically develop and test on one machine
and then deploy on another. For example,
ACTION="/servlet/schedule-fund-raiser"

III. CORE APPROACH

In addition, you can specify an email address to which the FORM data will be
sent (e.g., mailto:audit@irs.gov). Some ISPs do not allow ordinary users to create
server-side programs, or they charge extra for this privilege. In such a case,
sending the data by email is a convenient option when you create pages that
need to collect data but not return results (e.g., for accepting orders for products).
You must use the POST method (see METHOD in the following subsection)
when using a mailto URL.
Also, note that the ACTION attribute is not required for the FORM element. If
you omit ACTION, the form data is sent to the same URL as the form itself. See
Section 4.8 for an example of the use of this self-submission approach.
METHOD
The METHOD attribute specifies how the data will be transmitted to the HTTP
server. When GET is used, the data is appended to the end of the designated
URL after a question mark. For an example, see Section 19.1 (How HTML
Forms Transmit Data). GET is the default and is also the method that is used
when the user types a URL into the address bar or clicks on a hypertext link.
When POST is used, the data is sent on a separate line. Either GET or POST
could be preferable, depending on the situation.

Since GET data is part of the URL, the advantages of GET are that you can do
the following:

• Save the results of a form submission. For example, you can
submit data and bookmark the resultant URL, send it to a colleague
by email, or put it in a normal hypertext link. The ability to
bookmark the results page is the main reason google.com,
yahoo.com, and other search engines use GET.
• Type data in by hand. You can test servlets or JSP pages that use
GET simply by entering a URL with the appropriate data attached.
This ability is convenient during initial development.
Since POST data is not part of the URL, the advantages of POST are that you
can do the following:

Listing 1.3 MultipartForm.html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using ENCTYPE="multipart/form-data"</TITLE>

```
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H2>Using ENCTYPE="multipart/form-data"</H2>
<FORM
ACTION="http://localhost:8088/SomeProgram"
ENCTYPE="multipart/form-data"
METHOD="POST">
First name:
<INPUT TYPE="TEXT" NAME="firstName"
VALUE="Joe"><BR>
Last name:
<INPUT TYPE="TEXT" NAME="lastName"
VALUE="Hacker"><P>
<INPUT TYPE="SUBMIT">
</FORM>
</CENTER>
</BODY></HTML>
```

1.4 Push Buttons

Push buttons are used for two main purposes in the HTML forms: to submit forms and
to reset the controls to the values specified in the original HTML. Browsers that use
JavaScript can also use buttons for a third purpose: to trigger arbitrary JavaScript
code.
Traditionally, buttons have been created by the INPUT element used with a TYPE
attribute of SUBMIT, RESET, or BUTTON. In HTML 4.0, the BUTTON element was
introduced and is supported by Internet Explorer 6.0 and Netscape 7.0. This new
element lets you create buttons with multiline labels, images, font changes, and the
like. However, earlier browsers may not support the BUTTON element.

IV. SUBMIT BUTTONS

HTML Element: <INPUT TYPE="SUBMIT" ...> (No End Tag)
Attributes: NAME, VALUE, ONCLICK, ONDBLCLICK, ONFOCUS, ONBLUR
When a submit button is clicked i.e double clicked, the form is sent to the servlet or other server-sideprogram designated by the ACTION parameter of the FORM. Although the action can be triggered in other ways, such as the user clicking on an image map, most forms
have at least one submit button. Submit buttons, like other form controls, adopt the
look and feel of the client operating system, so will look slightly different on different
platforms.

```
<INPUT TYPE="SUBMIT">
```

1.5 Check Boxes and Radio Buttons

Check boxes and the radio buttons are useful controls for allowing the user to select
among a set of predefined choices. Although check boxes can be selected or deselected
individually, radio buttons can be grouped so that only a single member of the
group can be selected at a time.

V. CHECK BOXES

HTML Element: <INPUT TYPE="CHECKBOX" NAME="..." ...>
(No End Tag)
Attributes: NAME (required), VALUE, CHECKED, ONCLICK, ONFOCUS, ONBLUR
This input element creates the check box whose name/value pair is transmitted only if
the check box is checked when the form is submitted. For instance, the following
code results in the check box.

```
<P>
<INPUT TYPE="CHECKBOX" NAME="noEmail"
CHECKED>
Check here if you do <I>not</I> want to
get our email newsletter
```

VI. RADIO BUTTONS

HTML Element: <INPUT TYPE="RADIO" NAME="..."
VALUE="..." ...> (No End Tag)
Attributes: NAME (required), VALUE (required), CHECKED, ONCLICK,
ONFOCUS, ONBLUR

Radio buttons differ from the check boxes in that only a single radio button in a given
group can be selected at any one time. You indicate the group of radio buttons by providing

all of them with the same NAME. Only one button in a group can be depressed
at a time; selecting a new button when one is already selected results in the previous
choice becoming deselected. The value of the one selected is sent when the form is
submitted. Although radio buttons technically need not appear near to each other,
this proximity is almost always recommended.
An example of a radio button is shown in Listing 1.7. Because input elements are
wrapped as part of normal paragraphs, a DL list is used to make sure that the buttons
appear under each other in the resultant page and are indented from the heading above
them.In this case, creditCard=java would get sent
as part of the form data when the form is submitted.

Listing 1.7 Example of a radio button group

```
<DL>
<DT>Credit Card:
<DD><INPUT TYPE="RADIO"
NAME="creditCard" VALUE="visa">
Visa
<DD><INPUT TYPE="RADIO"
NAME="creditCard" VALUE="mastercard">
Master Card
<DD><INPUT TYPE="RADIO"
NAME="creditCard"
VALUE="java" CHECKED>
Java Smart Card
<DD><INPUT TYPE="RADIO"
NAME="creditCard" VALUE="amex">
American Express
<DD><INPUT TYPE="RADIO"
NAME="creditCard" VALUE="discover">
Discover
</DL>
```

19.6 Combo Boxes and List Boxes

A SELECT element presents a set of options to the user. If only a single entry can be
selected and no visible size has been specified, the options are presented in a combo
box (drop-down menu); list boxes are used when the multiple selections are permitted or
a specific visible size has been specified. The choices themselves are specified by
OPTION entries embedded in the SELECT element. The typical format is as follows:

```
<SELECT NAME="Name" ...>
<OPTION VALUE="Value1">Choice 1 Text
<OPTION VALUE="Value2">Choice 2 Text
...
<OPTION VALUE="ValueN">Choice N Text
</SELECT>
```

The HTML 4.0 specification also defines OPTGROUP (with a single attribute of
LABEL) to enclose OPTION elements to create cascading menus.
HTML Element: <SELECT NAME="..." ...> ... </SELECT>
Attributes: NAME (required), SIZE, MULTIPLE, ONCLICK, ONFOCUS, ONBLUR,
ONCHANGE
SELECT creates a combo box or list box for selecting among choices. You specify
each choice with an OPTION element enclosed between <SELECT ...> and
</SELECT>.

Listing 19.8 Example of a SELECT menu

```
Favorite language:
<SELECT NAME="language">
<OPTION VALUE="c">C
<OPTION VALUE="c++">C++
<OPTION VALUE="java" SELECTED>Java
<OPTION VALUE="lisp">Lisp
<OPTION VALUE="perl">Perl
<OPTION VALUE="smalltalk">Smalltalk
</SELECT>
```

REFERENCES

http://www.w3schools.com/