

25.1 Introduction

To execute a meaningful business transaction on the Internet, the client side (that is, the user's browser) has to be able to send information to the web server. This lesson provides a brief overview of different ways in which browsers can communicate with web servers.

25.1.1 Web 101

The Internet is just a **packet-switched** network that is capable of carrying all types of traffic including mail, chat, and proprietary data. The hypertextual, multimedia infrastructure we know as the World-Wide Web (WWW) is just one type of Internet traffic.

What gives the WWW its power is two high-level standards developed by Tim Berners-Lee in the early 1990s while working at CERN: HTML and HTTP. The HTML standard, which you saw in [Lesson 24](#), defines how documents are displayed within web browsers. The **hypertext transfer protocol** (HTTP) defines how the web server and the web browser communicate over the Internet in the first place. You can think of these protocols as being layered: HTML is “carried by” HTTP.¹

25.1.2 HTTP requests and responses

Whenever you click on a hyperlink on a web page, your browser creates an HTTP **request** and sends it out on the network. An HTTP request is simply text message that is routed to a particular web server. For example, consider the following HTTP request:

```
NL GET mis.bus.sfu.ca/Default.htm
HTTP/1.0
```

This **GET** messages asks a particular web server (**mis.bus.sfu.ca**) to send back a particular document (**Default.htm**). It also specifies the version of HTTP to avoid possible confusion as the protocol evolves.

Upon receipt of the request, the web server obliges by sending back an HTTP **response** containing the HTML contents of the requested page (in this example, **Default.htm**).

¹ Other lower-level protocols—such as the Transmission Control Protocol (TCP) and Internet Protocol (IP)—“carry” HTTP. However, to understand the basics, we can assume all the nitty-gritty network stuff is given and focus on the so-called “application layers”.



25.1.3 Sending additional data

Of course, a simple **GET** request is insufficient if you want to tell the web server who you are or that you wish to order product number "51 5012". Fortunately, the HTTP standard defines two basic mechanisms for passing additional data to the web server in the request: **query strings** and **form fields**. In this lesson, you will learn about both and get some experience creating simple HTML forms.

25.2 Learning objectives

- understand what an HTTP request is and how it is used to retrieve content from web servers
- pass information to the server using GET requests and query strings
- pass information to the server using POST requests and form fields
- learn how to create simple forms in HTML
- create more advanced form elements like radio buttons, check boxes and drop-down lists

25.3 Exercises

In the following exercises, you will send a number of HTTP requests to a special "Echo Request" utility. Echo Request is a small

program on a web server that extracts and displays ("echos") the information sent to it by clients.



The Echo Request document used in this lesson is located on a web server at Simon Fraser University (e-commerce.bus.sfu.ca). If you have your own web server software installed (see [Section 24.4.2](#)), you can copy the **EchoRequest.asp** file from the [project package](#) to a virtual directory on your local web server. You then use the URL of the local copy (`localhost/<virtual directory>/EchoRequest.asp`) in the place of `e-commerce.bus.sfu.ca`.

- ➔ Send an "empty" HTTP **GET** command to the Echo Request utility:

```
NL http://e-commerce.bus.sfu.ca/  
ASPTest/EchoRequest.asp
```

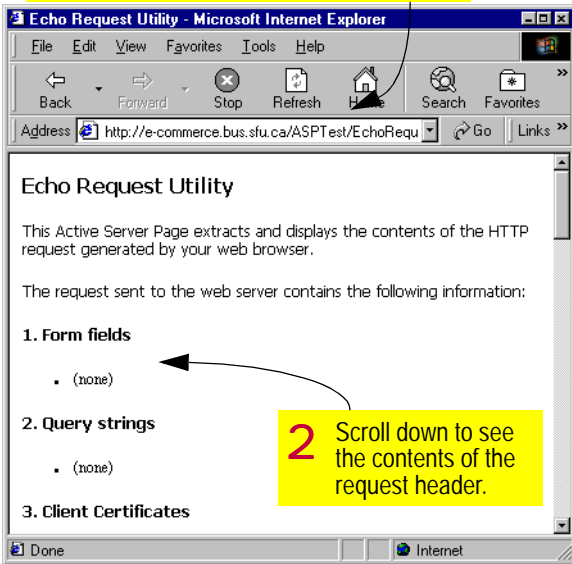
- ➔ Examine the information returned by the Echo Request utility, as shown in [Figure 25.1](#).



In addition to returning information about basic collections (form fields and query strings), the Echo Request utility returns information about the request's advanced collections, such as **Client Certificate** (for

FIGURE 25.1: Result of sending a simple **GET** command to the Echo Request utility.

1 The request sent to the server contains no query strings or form fields.



2 Scroll down to see the contents of the request header.

robust authentication), **Cookies**, and **Server Variables**. You can ignore this information for now. In this lesson, we are only interested in the Form and Query String collections.

25.3.1 Passing data using query strings

A query string is a quick and easy means of passing information to the web server. The information is passed to the server by appending one or more query/value pairs to the URL in the **GET** request.

➔ Enter the following URL and query string combination into your browser. The result is shown in **Figure 25.2**.

```
NL http://e-commerce.bus.sfu.ca/
ASPTTest/EchoRequest.asp?
UserName=sammy&Password=sam
```

A query string consists of one or more query/value pairs. In the example above, you send two such pairs to the server:

- UserName = "sammy"
- Password = "sam"

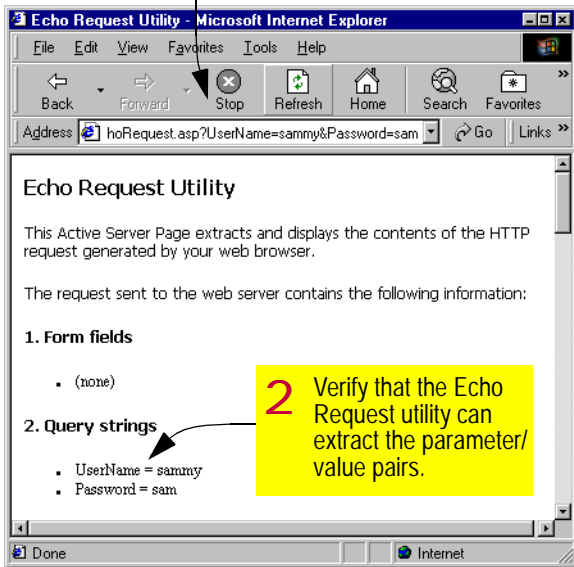
Note that the start of the query string is signified by a question mark (?). When multiple parameter/value pairs are passed, an ampersand (&) is used to separate the pairs.

❓ The name "query string" is unfortunate because passing data in this manner has little in common with the database concept of "queries". A better name might be "parameter string". However—as is often the case in computing—we are stuck with the legacy name.



FIGURE 25.2: Send a query string with two query/value pairs to the server.

- 1 Start the query string with a question mark (?) and separate query/value pairs with an ampersand (&). Do not add spaces.



Given the less-than-intuitive syntax of query strings, it is clear that they cannot be used for collecting information from users. For example, you would not expect users to append their user name and password to a URL in order to log on

to a secure system. In addition, the query string shows in the browser's address field. Because of this, query strings should not be used for information that users (or people standing behind users) do not need to see.

As it turns out, query strings are most often used by developers to pass status information to the web server in order to get around the "statelessness" of the HTTP protocol. You will see how this works in [Lesson 26](#).

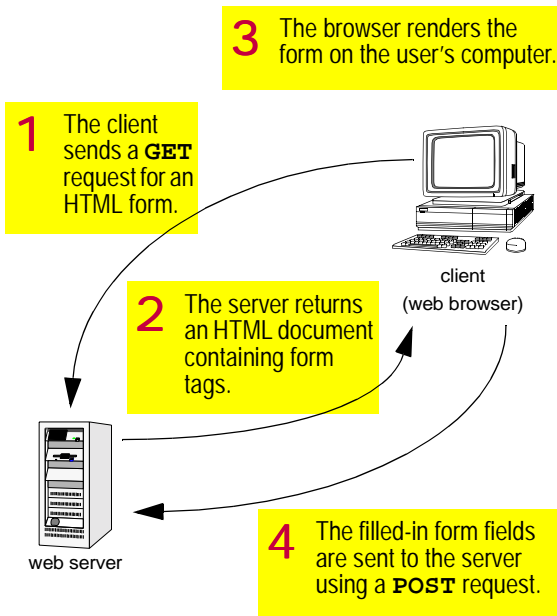
25.3.2 Passing data using forms

Fortunately, the HTTP and HTML standards provide mechanisms for displaying forms on the client browser and transferring the information entered by the user back to the web server. The key elements of the HTTP/HTML forms infrastructure are shown in [Figure 25.3](#).



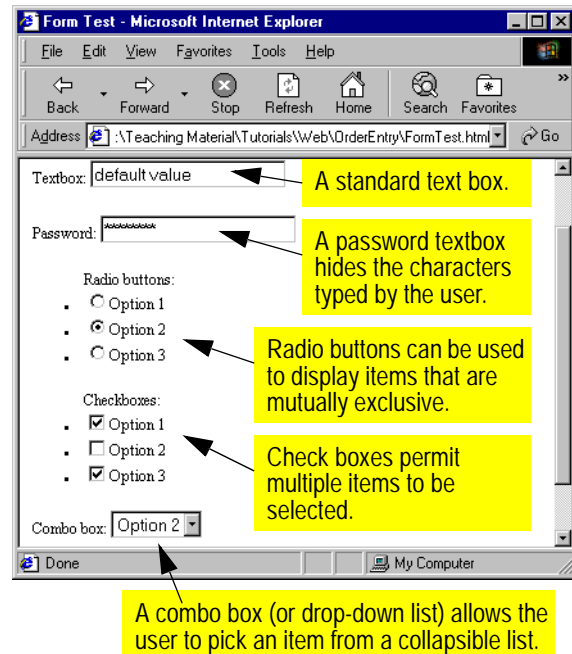
At this point, nothing has been said about what the web server *does* with the information that it receives from the user. Generally, this information is processed by a program running on the server. However, getting information from the web server to a server-side processing routine and back to the web server is not trivial. Indeed, it may involve many acronyms (e.g., CGI, ASP, CFML, PHP, JSP, ISAPI, ADO, COM, etc.). We will deal with some processing issues in later lessons.

FIGURE 25.3: Key elements of the HTML/ HTTP form infrastructure.



shows some of the form elements that can be defined using HTML.

FIGURE 25.4: HTML form elements rendered by a web browser.



An HTML form is like any other HTML document that resides on the web server. However, when the browser receives the form, it recognizes special HTML tags and renders the form elements on the user's screen. [Figure 25.4](#)

To define a form, you must provide the following:



- the type of HTTP request that is to be created by the browser and sent to the web server (the **METHOD** attribute),
- the destination of the HTTP request (the **ACTION** attribute), and
- a **SUBMIT** button to send the HTTP request.

In HTML, these elements can be added using the following tags:

```
NL <FORM METHOD="POST"
    ACTION=target URL>
NL   form elements ...
NL   <INPUT TYPE="submit" VALUE=label>
NL </FORM>
```



There are two possible values for the **METHOD** attribute: **GET** and **POST**. All you need to know at this point is that **POST** is almost always used with forms.

- ➔ Create a new HTML document with header and body sections. Save it as **FormTest.html**.
- ➔ In the body section, enter the following:

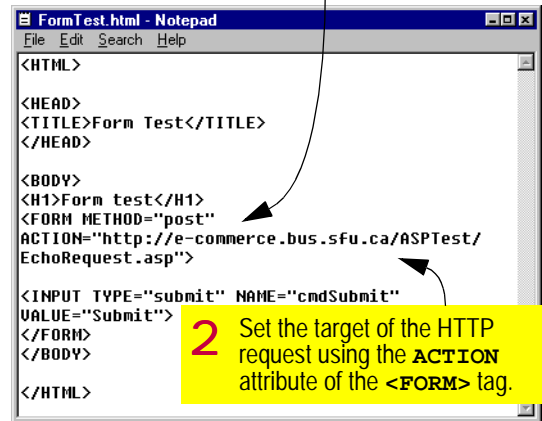
```
NL <BODY>
NL <H1>Form test</H1>
NL <FORM METHOD="post" ACTION="http://
    e-commerce.bus.sfu.ca/ASPTTest/
    EchoRequest.asp">
NL <INPUT TYPE=submit NAME="cmdSubmit"
    VALUE="Submit">
```

```
NL </FORM>
NL </BODY>
```

- ➔ Save the document. The HTML code is shown in [Figure 25.5](#).

FIGURE 25.5: Create a basic HTML form consisting of a **Submit** button.

- 1 Create a simple form with a single form element: a **Submit** button.



- 2 Set the target of the HTTP request using the **ACTION** attribute of the **<FORM>** tag.

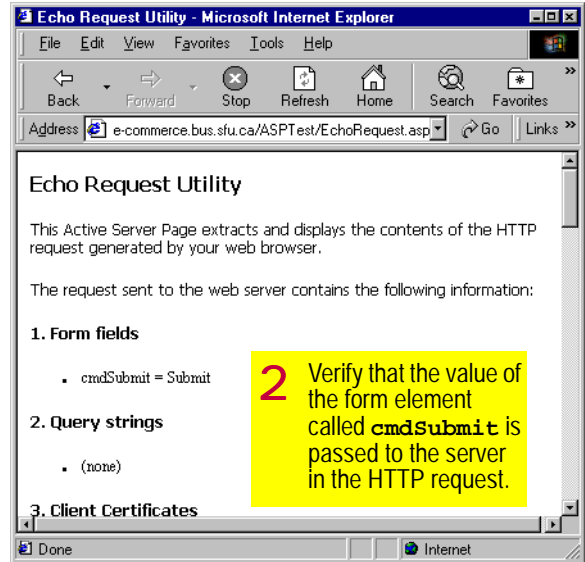
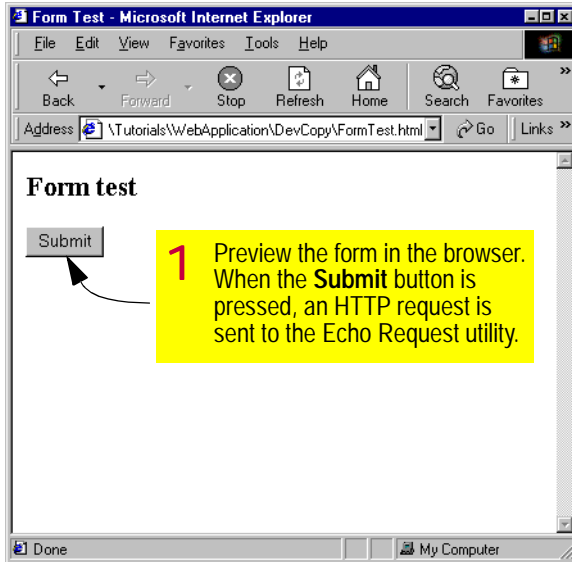


Remember: whitespace is ignored in HTML and thus the wrapping of your lines within the editor is irrelevant.

➔ Test the form using a web browser. When you press the **Submit** button, the Echo Response utility will show you the contents

of the HTTP request, as shown in Figure 25.6.

FIGURE 25.6: Create a basic HTML form and test it by sending the HTTP response to the Echo Response utility.



25.3.3 Basic form elements

A form that simply sends the value of the **Submit** button is not particularly useful. Elements that are typically included on forms

include the textbox and its cousin, the password textbox.



- ➔ Add the following to your HTML document (after the `<FORM>` tag but before the **Submit** button):

```
NL <BODY>
NL <H1>Form test</H1>
NL <FORM METHOD="post" ACTION="http://
e-commerce.bus.sfu.ca/ASPTest/
EchoRequest.asp">
NL <P>Textbox: <INPUT TYPE="text"
NAME="txtItem" VALUE="default
value"></P>
NL <INPUT TYPE=submit NAME="cmdSubmit"
VALUE="Submit">
NL </FORM>
NL </BODY>
```



Make sure you remember the quotation marks around the attribute values. Otherwise, browsers will misinterpret "default value".

- ➔ Under the textbox you added above, add a password textbox:

```
NL <P>Password: <INPUT TYPE="password"
NAME="txtPassword"></P>
```



The form elements above are nested inside of paragraph tags to space things out a bit; however, you may choose to format your page however you like. In general, it is very difficult to get form elements to line up nicely. As such, web

designers routinely place form elements inside of invisible table cells or use other formatting tricks. At this stage, you should not worry about how your forms look.

- ➔ Save the file and refresh the form in your browser.
- ➔ Press the **Submit** button and verify that the values you typed were sent to the web server. This is shown in [Figure 25.7](#).



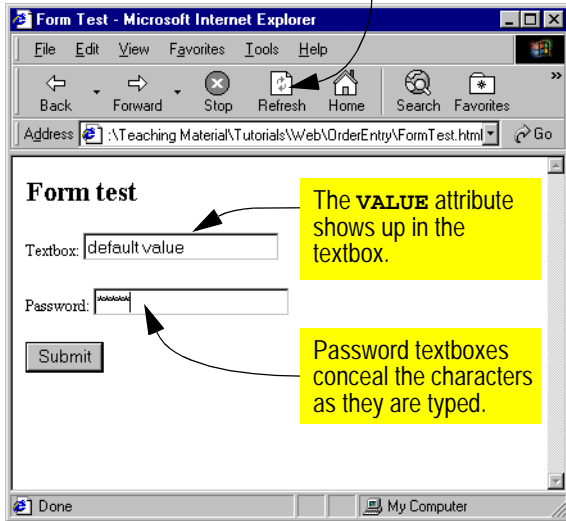
Although the value in the password textbox is concealed by the web browser, this feature is merely intended to hide the value from other people who may be able to see the web browser. There is no protection or encryption once the **Submit** button is pressed—the password travels across the network as plain text. See [Section 25.4](#) for more information on security and encryption.

25.3.4 Other form elements

All the form elements created so far use the `<INPUT>` tag with different values for the **TYPE**

FIGURE 25.7: Add a textbox and password textbox to the form.

1 Press the **Refresh** button to show the latest changes to the HTML document.



attribute. The possible types of form elements are summarized below.

TABLE 25.1: Form elements in HTML.

TYPE value	Usage
text	simple textboxes for names, etc.

TABLE 25.1: Form elements in HTML.

TYPE value	Usage
password	concealed text for confidential information such as passwords
submit	a button required to trigger the creation of an HTTP request and send it to a web server
reset	also rendered as a button; resets all form elements to the values specified in their VALUE attributes (if any)
radio	creates a group of radio buttons of which <i>only one</i> can be selected (mutually exclusive options)
checkbox	creates one or more checkboxes of which <i>zero or more</i> can be checked (non-mutually exclusive options)
button	creates a button the form that is not assigned any behavior by default
hidden	the element's value is transferred to the server, but no field is shown on the form; can be used instead of query strings to transfer status information

The other attributes of the **<INPUT>** tag, **NAME** and **VALUE**, can take on any values specified by the creator of the form.



As in all programming endeavors, it is good policy to adopt a meaningful and consistent naming policy for HTML elements.

- ➡ Add a group of radio buttons to your form:

```
NL ...
NL <BODY>
NL ...
NL <P>Password: <INPUT TYPE="password"
NL NAME="txtPassword"></P>
NL <UL>Radio buttons:
NL <LI><INPUT TYPE="radio"
NL NAME="rdoGroup" VALUE="opt1">Option
NL 1</LI>
NL <LI><INPUT TYPE="radio"
NL NAME="rdoGroup" VALUE="opt2">Option
NL 2</LI>
NL <LI><INPUT TYPE="radio"
NL NAME="rdoGroup" VALUE="opt3">Option
NL 3</LI>
NL </UL>
NL <INPUT TYPE=submit NAME="cmdSubmit"
NL VALUE="Submit">
NL </FORM>
NL </BODY>
NL ...
```



`` is a standard HTML tag used to define an “unordered list” (an “ordered list” uses the `` tag and has numbers instead of bullets). Each “list item” in the list is defined using the `...`

tags. Of course, it is not necessary to nest radio buttons within a list—you may format them any way you wish.

- ➡ Save the document and test the radio buttons by selecting one of the options.
- ➡ Send the HTTP request to the server and observe the result.

Note that all the radio buttons are assigned the same name yet only one value is assigned to the `rdoGroup` field. A radio button will always return a single value since the browser prevents the user from selecting more than one option. The value of `rdoGroup` is simply the `VALUE` attribute of the radio button that is selected.

Checkboxes are similar to radio buttons, except that the former permit multiple values to be assigned to a single form field. Checkboxes are often used for on-line customer surveys. If respondents are asked, “which product features are important?” and more that one answer is possible, multiple checkboxes can capture a multivalued response, e.g., `chkFeatures = {price, quality, service, reputation}`

- ➡ Add a group of checkboxes to your form:

```
NL ...
NL <BODY>
NL ...
```



```

NL <LI><INPUT TYPE="radio"
NAME="rdoGroup" VALUE="opt3">Option
3</LI>
NL </UL>
NL <UL>Checkboxes:
NL <LI><INPUT TYPE="checkbox"
NAME="chkGroup" VALUE="opt1">Option
1</LI>
NL <LI><INPUT TYPE="checkbox"
NAME="chkGroup" VALUE="opt2">Option
2</LI>
NL <LI><INPUT TYPE="checkbox"
NAME="chkGroup" VALUE="opt3">Option
3</LI>
NL </UL>
NL <INPUT TYPE=submit NAME="cmdSubmit"
VALUE="Submit">
NL </FORM>
NL </BODY>
NL ...

```

- ➔ Save the document and test the checkboxes by selecting more than one of the options. The results are shown in [Figure 25.8](#).

25.3.5 Combo boxes

It is possible to create combo boxes similar to those created in ACCESS in [Lesson 15](#).



“Combo box” is a MICROSOFT term but it is used here for consistency with the ACCESS material. In the Internet world, such form

elements are typically called “drop-down lists” or “list boxes”.

The combo box itself is defined by the `<SELECT>` tag. The list that shows when the combo box is activated is defined by one or more `<OPTION>... </OPTION>` tags:

- the **VALUE** attribute within the option tag determines the value that is returned if the option is selected by the user;
- the text within the opening and closing tags determines what shows in the combo box.

For example, the following tag adds the value “Option 1” to the list of items in a combo box called “cboItems”:

```

NL <SELECT NAME="cboItems">
NL <OPTION VALUE="opt1">Option 1</
OPTION>
NL ... other list items...
NL </SELECT>

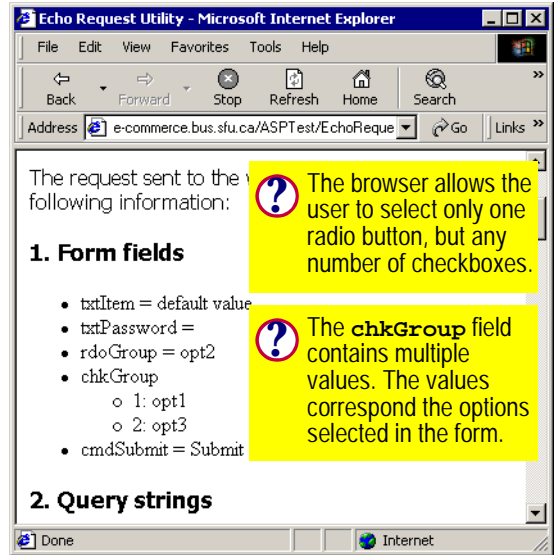
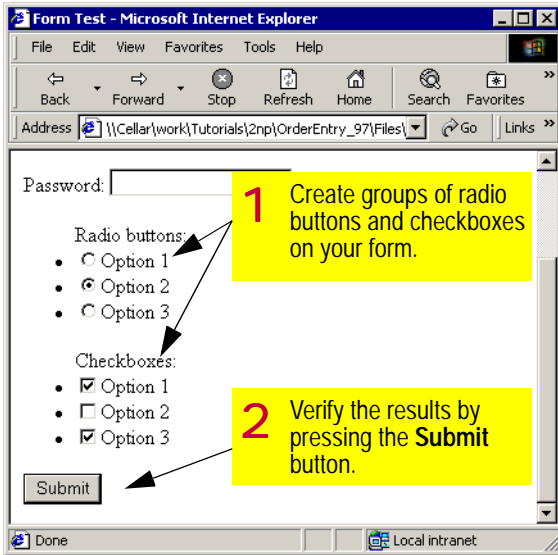
```

If the user selects the value “Option 1” from the combo box, then the value of `cboItems` is set to “opt1”. In other words, HTML provides a means of hiding the key in the `<OPTION>` tag’s **VALUE** attribute while showing the user a more meaningful value in the visible list (recall using ACCESS to do the same thing in [Section 15.3.2.4](#)).

- ➔ Add a combo box to your form:

```
NL <BODY>
```

FIGURE 25.8: Create radio button and checkbox groups on the form.

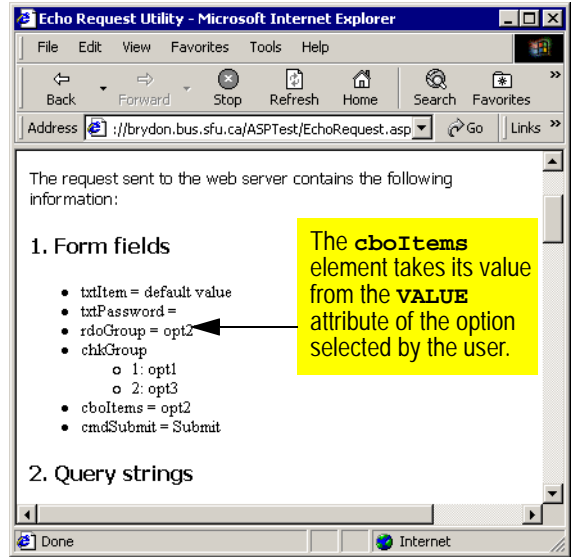
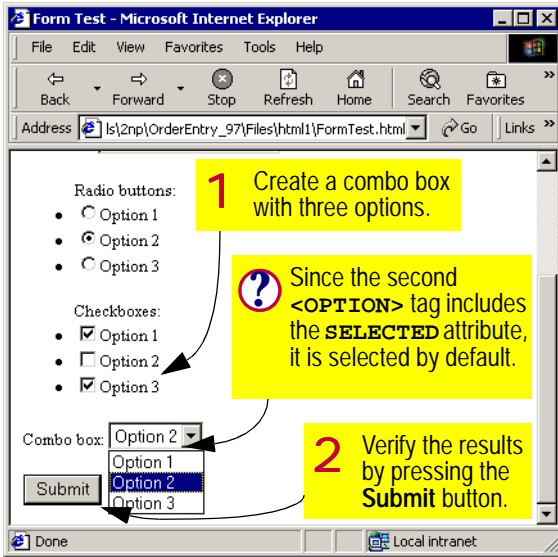


```
NL ...
NL <LI><INPUT TYPE="checkbox"
NAME="chkGroup" VALUE="opt3">Option
3</LI>
NL </UL>
NL <P>Combo box:
NL <SELECT NAME="cboItems">
NL <OPTION VALUE="opt1">Option 1</
OPTION>
NL <OPTION VALUE="opt2"
SELECTED>Option 2</OPTION>
```

```
NL <OPTION VALUE="opt3">Option 3</
OPTION>
NL </SELECT></P>
NL <INPUT TYPE="submit"
NAME="cmdSubmit" VALUE="Submit">
NL </FORM>
NL </BODY>
```

➔ Save the document and test the combo box. The result is shown in [Figure 25.9](#).

FIGURE 25.9: Create a combo box (dropdown list) on the form.



25.3.6 Menu forms

Customers will use your application's main menu to select the pages they wish to view. For example, they may wish to update their customer profile and then logoff. Alternatively, they may need to view an existing order and then create a new order. A simple menu for navigating from page to page (e.g., from

`Customer.html` to `Logoff.html`) is easy to create using hyperlinks (recall [Section 24.3.3](#)). However, if the content of the target page is variable (e.g., one may jump to a new order or a particular order created in the past), then a form-based menu provides greater flexibility.



25.3.6.1 Multiple submit buttons

- ➔ Open `Menu.html` for editing and add form definition tags to the body section. At this point, set the `ACTION` attribute to the Echo Request utility.
- ➔ Add a menu item and a separate **Submit** button for each of the choices available to the user. An invisible two-column table can be used to simplify the positioning of the menu items and buttons.

```
NL <HTML>
NL <HEAD>... </HEAD>
NL <BODY>
NL ...
NL <FORM METHOD="POST" ACTION="http://
e-commerce.bus.sfu.ca/
ASPTest/EchoRequest.asp">
NL <TABLE>
NL <TR>
NL <TD>Update Customer Profile</TD>
NL <TD><INPUT TYPE="Submit"
NAME="cmdCustomer" VALUE="Go"></TD>
NL </TR>
NL <TR>
NL <TD>View Product List</TD>
NL <TD><INPUT TYPE="Submit"
NAME="cmdProduct" VALUE="Go"></TD>
NL </TR>
NL <TR>
NL <TD>Add or View Orders</TD>
```

```
NL <TD><INPUT TYPE="Submit"
NAME="cmdOrder" VALUE="Go"></TD>
NL </TR>
NL ...
NL </TABLE>
NL </FORM>
NL </BODY>
NL <HTML>
```



Use a different descriptive name for each button (e.g., `cmdCustomer`, `cmdProduct`, `cmdOrder`, `cmdLogoff`).

- ➔ Save the file and view it in the browser, as shown in [Figure 25.10](#).
- ➔ Test the menu by clicking on different **Go** buttons. As the Echo Request utility shows in [Figure 25.11](#), the name/value pair for the button that is pressed—and *only* the button that is pressed—is transferred to the server.
- ❓ In a subsequent lesson, you will write a server-side script to determine which button was pressed and transfer the user to the appropriate page. For now, creating the menu items and buttons is sufficient.

25.3.6.2 Menu items and combo boxes

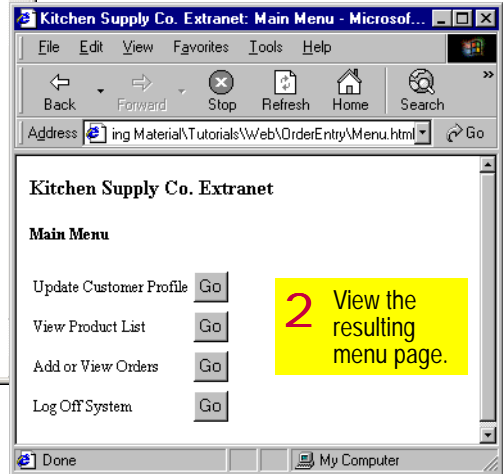
The menu in [Figure 25.10](#) is incomplete because the user has no means of indicating whether she

FIGURE 25.10: Create a form-based menu for the application.

```

menu.html - Notepad
File Edit Format Help
<H2>Kitchen Supply Co. Extranet: Main Menu</H2>
<FORM METHOD="post"
ACTION="http://e-commerce.bus.sfu.ca/ASPTest/EchoRequest.asp">
<TABLE>
  <TR>
    <TD>Update Customer Profile</TD>
    <TD><INPUT TYPE="Submit" NAME="cmdCustomer" VALUE="Go"></TD>
  </TR>
  <TR>
    <TD>View Product List</TD>
    <TD><INPUT TYPE="Submit" NAME="cmdProduct" VALUE="Go"></TD>
  </TR>
  <TR>
    <TD>Add or View Orders</TD>
    <TD><INPUT TYPE="Submit" NAME="cmdOrder" VALUE="Go"></TD>
  </TR>
  <TR>
    <TD>Log Off System</TD>
    <TD><INPUT TYPE="Submit" NAME="cmdLogoff" VALUE="Go"></TD>
  </TR>
</TABLE>
</FORM>
  
```

1 Create a form to transfer different values to the web server depending on which **Submit** button is pressed.



? The ACTION attribute should be set to the name of the web server that you are using.

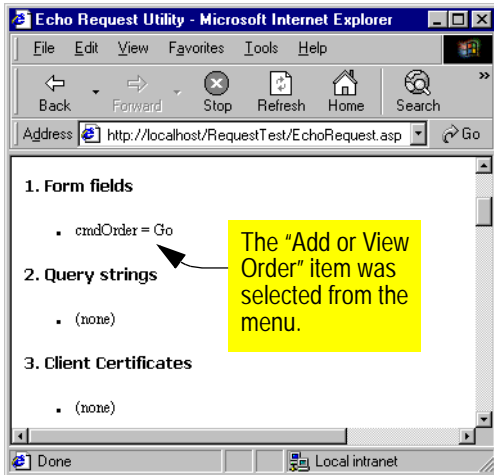
wants to create a new order or view an existing order (i.e., an order that has already been processed or one that has been started by not yet finalized). If the user wants to view an existing order, then she must have some way of telling the system *which* order (remember, a customer may have placed many different orders in the past).

To correct this shortcoming, you can add a combo box that lists all the existing orders (using some compact naming convention, such as order date) plus an option for creating an entirely new order.

➔ In the "Add or View Orders" table cell, add the following code to create a combo box:



FIGURE 25.11: The name/value pair of the selected item is transferred to the server.



```
NL <FORM METHOD="POST" ACTION="http://
e-commerce.bus.sfu.ca/ASPTest/
EchoRequest.asp">
NL <TABLE>
NL ...
NL <TR>
NL <TD>Add or View Orders
NL <SELECT NAME="cboOrderID">
NL <OPTION VALUE=0>(new order)</
OPTION>
NL <OPTION VALUE=1>15 May</OPTION>
```

```
NL <OPTION VALUE=2>21 May</OPTION>
NL </SELECT>
NL </TD>
NL <TD><INPUT TYPE="Submit"
NAME="cmdOrder" VALUE="Go"></TD>
NL </TR>
NL ...
NL </TABLE>
NL </FORM>
```



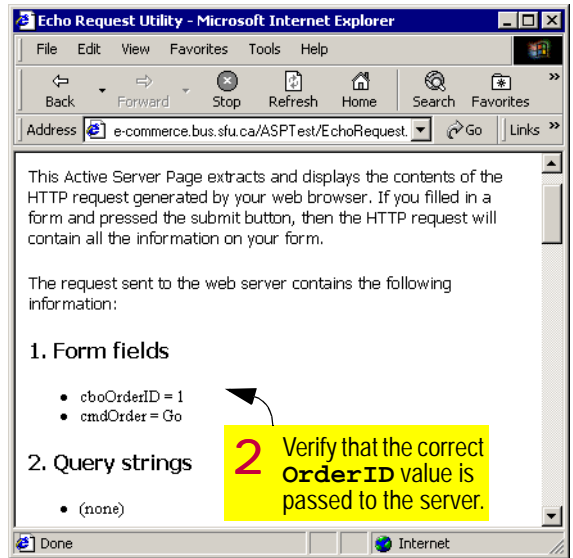
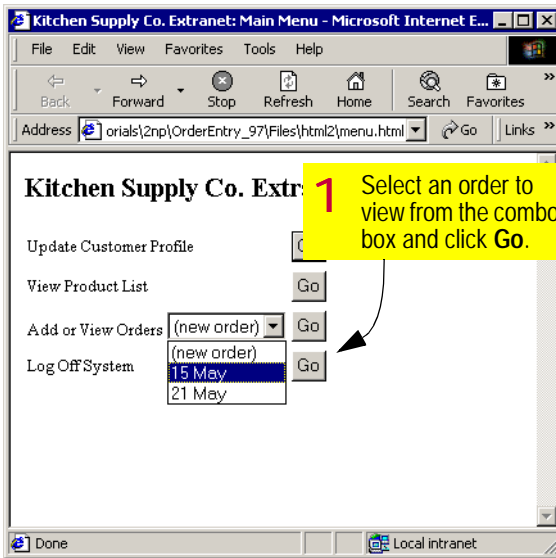
I have added the combo box to the same table cell (that is, within the same `<TD>...</TD>` tags) as the prompt "Add or View Orders". For this to work, you must remember to delete the closing `</TD>` tag at the end of the prompt and add a new one at the end of the `</SELECT>` tag. Of course, you can format your menu page anyway you like.

This combo box above assumes that two orders (with `orderIDs` of 1 and 2) have already been entered into the system. Since it is not possible to have an `orderID` of zero, the 0 option value is used to indicate a new order.

➔ Test the menu by selecting a value from the combo box and clicking the appropriate **Go** button. One possible outcome is shown in [Figure 25.12](#).

By using a form to create the main menu, you are able to pass the server more information

FIGURE 25.12: Add a combo box to the menu to specify which order to view or update.



than if a simply hyperlink were used. In Figure 25.12 for instance, the server is sent two pieces of information:

1. The user wishes to view an order (`cmdOrder=Go`).
2. The order that the user wishes to view has `OrderID=2`.

25.4 Discussion

25.4.1 Security concerns

In Figure 25.7, you sent a password over the public Internet to a web server. In travelling to the web server, the IP data packets carrying the HTTP request could have been routed through many different sub-networks and computers. Since HTTP is a plain-text protocol, all field/



value pairs are easily read by so-called “packet sniffers”. A packet sniffer is a program that can extract data from the stream of IP packets travelling along a network.

Although reading all the network traffic passing through a router or server would involve a fair bit of sniffing, it is fairly straightforward to create a program that watches for field names such as “`txtCreditCardNo`” or the characteristic sequences of digits that prefix credit card numbers.

25.4.2 Encryption

To get around the problem, most confidential information is now encrypted by the browser, sent over the network, and decrypted by the target web server. A packet sniffer looking at the contents of the HTTP request would only see a jumble of characters (“cipher text”) in the place of well-defined field/value pairs.

There are different standards for encryption and authentication, but the most common on the Internet at this time is the secure sockets layer (SSL). Although it is possible—in principle—to decrypt streams of encrypted data, it makes little economic sense in practice to even try.

25.5 Application to the project

➔ Add a form to your login page and set its **ACTION** attribute to point to the Echo Request utility (this will be changed later).

➔ Add a textbox called `txtUserName`.



In [Section 25.3.3](#), you created a textbox with the **VALUE** attribute preset to “default value”. The user could replace the initial value by typing something into the form field. It is important to note, however, that providing an initial value to the **VALUE** attribute is optional. Indeed, in the context of a textbox for a user name, use of a default value makes little sense (unless you can somehow guess who the user is before she visits your site).

➔ Add a password textbox called `txtPassword`.

➔ Add a submit button called `cmdSubmit`. Its **VALUE** attribute should be set to “Submit”.



