In addition to some minor display message changes, a new `div` tag with an ID of `AjaxReturnValue` has been created to hold the dogs list box (which will provide the user a selection of dogs to choose). An **HTML button** (not a Submit button) follows. When clicked, it will cause a JavaScript function (`process_select`) to execute. Another `div` tag has been added which contains the HTML form. This form is to be hidden until the user clicks the button.

CSS code (`#input_form { display:none; }`) is included at the top of the program to keep the form from displaying. Additional CSS code also keeps the buttons from displaying. This code is very similar to the CSS code to keep the non-JS form from displaying if the user does not have JavaScript activated in the browser.

At the bottom of the form, a hidden property (`index`) has been created to hold the index (from the dogs array) of the dog selected. The initial value is set to `-1`. This property will be changed when the user selects a dog. The original Submit button is replaced by three Submit buttons (one for insert, one for delete, and one for update). Whichever button is clicked will cause a property to be created (`insert`, `delete`, or `update`) and set to a value. The property name is the ID of the button, and the contents in the property are the contents of the `value` attribute of the button selected. This will help `dog_interface` determine which type of changes the user is requesting. These buttons are also included on the non-JavaScript enabled form. Remember, in this example, non-JavaScript enabled browsers will be require users to enter all information needed to successfully accomplish an `insert`, `delete`, or `update`.

Hopefully, the changes you just looked at are pretty understandable. You are now going to look at some JavaScript code to handle this data. As mentioned, the manipulation of data by JavaScript is a common task in web applications. While this is not a JavaScript book, it is important that any web applications developer be familiar with JavaScript. I think you will see that the structure of the JavaScript language is similar to the structure of the PHP language.

First you will look at the changes to the `get_breeds.js` file (from Chapter 4). The file is renamed to `getlists.js` to reflect that it will now handle the `getBreeds` and `dogs` list boxes.

```
function HandleResponse(response)
{
        var responsevalues = response.split('|');
        document.getElementById('AjaxResponse').innerHTML = responsevalues[0];
        document.getElementById('AjaxReturnValue').innerHTML = responsevalues[1];
        obj = JSON.parse(responsevalues[2]);
}
```

All the code changes are in the `HandleResponse` method of the JavaScript file. Previously the values in the response property (passed to the method) were directly passed into the `div` tag with the `AjaxResponse` ID. At that point, only the list box code for the breeds was returned. Now the method will accept three types of information (the breeds list box, the dogs list box, and the dogs array). To reduce the number of calls to the web server, one AJAX call is made. It returns all the information into the `response` property. The information will be separated by using the pipeline (`|`) symbol. Soon you will see that the formation of this string will occur in the `dog_interface` program.

You will need to break the data in a similar way that you broke previous data using the PHP `explode` method. In JavaScript, the `split` method will break a string into an array using a parameter provided (`|`). In the example, this will create the array responsevalues. `var` creates this array as local to the method. It will be destroyed, because it will no longer be needed, when the method closes (hits the } symbol). The array now has three rows. The first row (`[0]`) contains the `getBreeds` list box code. The second row (`[1]`) contains the `dogs` list box code. The third row (I bet you guessed that one) contains the complete dogs array.