We have probably read just about every book written on ASP.NET. Some are good, detail-oriented books while some others are merely introductory books that only touch key topics on the surface. Regardless of the level of depth these books provide, most of them don't educate readers on ways of writing client-side JavaScript.

Though these books cover lots of information on ASP.NET, they don't give a straightforward guide on how to check for a valid input value. You don't necessarily want to make a round trip to the server and take that hefty performance penalty just to check for a valid input value; you can do that on the client in a few milliseconds. We have had the JavaScript versus post-back conversation at least a dozen times with a variety of developers from a wide range of industries. Being consultants enables us to work with a diverse range of clients, helping them understand the impact of the ASP.NET technology. Soon after we get into the details of the post-back mechanism, some insightful programmer in the room always questions how ASP.NET supports JavaScript.

It's time to start educating programmers on how effectively to use JavaScript with the ASP.NET post-back mechanism to provide an efficient and reliable experience to the users of our Web applications. Even the creators of ASP.NET know this. The validation controls that ship with the framework make good use of client-side JavaScript to validate user input without having to make a round trip to the server.

We believe that every project team needs to decide on the amount of JavaScript they want to develop and support. JavaScript is not necessarily an easy language to work with. It provides virtually no support for advance debugging. It may behave differently on different browsers and on different versions of the same browser. When your project team decides to develop parts of the application by using JavaScript, it needs to be mentally and technically prepared to deal with its implications.

JavaScript can not only boost the application performance but can also provide a seamless and efficient experience to the users, specially those who use a slow Internet connection to browse your Web site.

For example, suppose you are asked to provide a drop-down list of credit card numbers, and, when the user selects a number, to display the cardholder name and expiration date by using text boxes. One way to accomplish this task is
to capture the SelectedIndexChanged event of the ASP.NET drop-down list control. When this event occurs, the page makes a post-back trip to the server and triggers your event handler. In the event handler, you can obtain the selected credit card number and retrieve remaining information by querying the database. This mechanism, even though it is an ASP.NET way of accomplishing tasks, is very slow and causes unnecessary traffic on the Web server.

This chapter shows an alternative approach to server-side event processing. It focuses on showing various aspects of using JavaScript with ASP.NET code to provide a rich and seamless user experience. When you are done reading this chapter, you will be able to create a JavaScript array dynamically and call a JavaScript function when the user selects an item from the list. By using this function, you can retrieve remaining information by searching the array. The JavaScript approach will be extremely fast and will not create an unnecessary load on the Web server.

Capturing Events Raised by HTML Controls with JavaScript

A significant feature of the JavaScript language is its ability to capture events fired by HTML controls that reside on the page. Notice that we used the term HTML controls instead of Web controls. Web controls are ASP.NET server controls that exist only on the Web server and perform all their functions while residing on the server. They are not available to the JavaScript code that runs in the browser and works on the HTML received from the Web server.

Prior to ASP.NET, receiving events from HTML controls was quite simple. Because Web controls didn't exist, the Web server directly manipulated HTML tags. In the past, when we needed to capture such event, we would simply use the JavaScript syntax for capturing such events and provide either inline JavaScript code or call a JavaScript function. The following example shows how events are captured in JavaScript:

```html
<input type="button" value="Hello" id="MyButton"
onClick="javascript: alert('Hi')">
```

When you place the preceding line of code in any HTML page, you will see a button with the text “Hello” on it. When you click this button, a JavaScript alert box that says “Hi” will appear. Accomplishing this task in ASP.NET is not as simple as it probably should be. If you need to create a button control in ASP.NET, chances are that you want a Web server control so that you can still capture post-back events and perform tasks on the Web server. In addition, if you also want to capture a client-side event by using JavaScript, you can't simply use the