Introduction to Ajax on ASP.NET

Over the years, we developers have seen many changes in terms of how development occurs. We have gone from terminal-based programming to PC-based programming to Windows-based programming to the web. Now we are on the verge of another programming revolution. This programming revolution will bring more interactive user interfaces to Web applications. This programming revolution is brought to developers courtesy of a set of technologies that are generally known as Ajax (Asynchronous JavaScript And XML). No longer will users see the annoying flash with the click of a button to submit data. No longer will users lose the context where they are located and be thrown back to the top of a page. With Ajax, developers can build applications that step out of the traditional postback model of the web, provide an improved user interface to users, and allow developers to develop applications that are much more user-friendly.

In this chapter, you are going to take a look at:

- ❑ ASP.NET development and how it led to Ajax
- ❑ What Ajax is and a high-level overview of some of its base technologies
- ❑ The advantages of Ajax
- ❑ Some things that it might not make sense to do with Ajax

Development Trends

If you have been developing for a while, like us old guys, you have gone through several iterations of development. Development has gone from terminals connected to mainframes and minicomputers to personal computers and then to client-server development. Client-server development allowed for the minimization of back-end resources, network resources, and the front-end PC by sending only the necessary data between back end and front end. Intelligent client-server development allowed for building applications that were responsive to the user and made efficient use of network and back-end resources. As the web development methodology took off in the late 1990s, we unfortunately returned to terminal-style development. In this methodology, any major operation between the client and server requires that all data be sent in what is called a *round trip*. With a round trip, all data from the form is sent from the client to the web server. The web server processes data, and then sends it back to the client. The result of a round trip is that lots of data is sent back and forth between the client and server. For example, form data, viewstate, and images may be sent back and forth without the need to be sent back and forth. Figure 1-1 shows how only a web browser is required at the client and how communications work with the web server being an intermediary between the client and any resources.

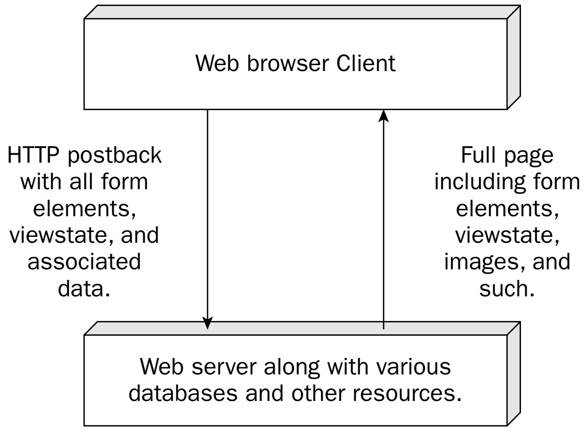
Web browser Client
HTTP postback with all form elements, viewstate, and associated data.
Full page including form elements, viewstate, images, and such.
Web server along with various databases and other resources.

Figure 1-1

ASP.NET Development

ASP.NET is a set of web development technologies produced by Microsoft that is used to build dynamic web sites, web applications, and XML-based web applications. ASP.NET is a part of the .NET framework and allows developers to build applications in multiple languages, such as Visual Basic .NET, Jscript, and C#.

Design Methodology

ASP.NET attempts to make the web development methodology like the graphical user interface (GUI) development methodology by allowing developers to build pages made up of controls similar to a GUI. A server control in ASP.NET functions similarly to GUI controls in other environments. Buttons, textboxes, labels, and datagrids have properties that can be modified and expose events that may be processed. The ASP.NET server controls know how to display their content in an HTML page just like GUI-based user controls know how to display themselves in their GUI environment. An added benefit of ASP.NET is that the properties and methods of the web server controls are similar, and in some cases the same as, those of comparable controls in the Windows GUI/Winforms environment.

Problems ASP.NET Solves

Microsoft has released various web application development methodologies over the past 10 years. Why do developers need ASP.NET? What problems does ASP.NET solve that the previous development methodologies did not solve?

Microsoft's first popular web development technology was the Internet Database Connector (IDC). The IDC methodology provided only database access; it did not provide access to any other resource programmatically. There was no way to programmatically send email or do other nondatabase operations. Another issue was that it seemed to be somewhat different from the traditional programming languages that most developers were used to (Visual Basic and C++ being two popular ones). Along with this problem was the fact that the development experience was not very attractive within Microsoft FrontPage. Along with the development experience, IDC had no debugging experience worth mentioning. Overall, IDC was nothing more than a stopgap measure to get to an improved environment.

The next web development methodology from Microsoft was Active Server Pages (ASP). ASP was a scripting environment that allowed developers to work with a Visual Basic like or JavaScript type environment. Unfortunately, this type of environment came with several problems:

- **Prevalence of spaghetti code** — ASP code does not provide a structured development environment, often contributing to the creation of twisted and tangled "spaghetti code." ASP code is literally a file with some basic configuration information at the top of every page. Each page is executed from the top of the page to the bottom of the page. While it is possible to use Component Object Model (COM) objects to eliminate some of the spaghetti code, this introduces more complexity in the form of another development tool.
- **Lack of code separation** — The code tends to be intermixed with display code. Intermixing the code and the display logic requires that the tools that developers and designers use work well together. This was often not the case. For example, it was well known that various visual development tools could take a properly running ASP page, rearrange some of the code, and render the ASP page broken.
- **Lack of code reusability** — There is very little ability to reuse code within the ASP environment.
- **Lack of debugging support** — Debugging an ASP application typically involves the use of `Response.Write`. This is in sharp contrast to an integrated development environment (IDE) developed within a GUI environment.
- **Problems of COM** — ASP is based on the Component Object Model and suffers from many of the problems associated with COM. There were two major problems with COM.
    - The first was that updating COM objects tended to overwrite one object with the new one. This could be problematic if a programming method call changed or any other new behavior was introduced.
    - The second major problem with COM was that it was a binary standard. This binary standard was based on a 32-bit programming model. As a result, COM objects would not scale up to run natively within an environment that was an Intel-based 32-bit environment. While this might not have been a big deal in the early to middle 1990s when COM was designed and built, by the early 2000s and the introduction of inexpensive 64-bit systems, this was seen as a possible bottleneck.
- **Problems with being interpreted** — ASP is interpreted. Each time an ASP file is loaded, the ASP environment parses the ASP file, compiles the code, and then executes the file. This process is repeated on each call to an ASP file. The result is wasted processing on the server.
- **Presence of the statemachine** — ASP applications typically have a statemachine (in software code, a statemachine is a section of code that depends on both its direct inputs and inputs made during previous calls) at the top of every ASP page that processes the state of the user and then displays code. Given that most client-side applications are built based on events, which is a similar concept to a statemachine, this is an unfamiliar way to develop for developers not well versed in ASP.

Upon discovering these problems, Microsoft developed ASP.NET. ASP.NET greatly simplifies the web development methodology.

- ❑ Developers no longer need to worry about processing state. With ASP.NET, actions are performed within a series of events that provide statemachine-like functionality.
- ❑ With the use of code-behind/beside model, code is separated from display. By separating code and display files, there is less of a chance of designer and developer tools interfering with each other.
- ❑ A single development tool may be used for building the application and business logic. Having a single integrated development suite allows developers to more easily interact with the application logic. This results in more code reuse and fewer errors.
- ❑ With the Visual Studio .NET IDE, ASP.NET supports many methods to debug and track a running ASP.NET.
- ❑ Because ASP.NET is based on the common language runtime (CLR) and .NET, ASP.NET does not suffer from the problems of COM. The .NET framework allows for multiple versions of components to be on a system without interacting with each other.
- ❑ ASP.NET is compiled. The first time that a file is loaded, it is compiled and then processed. The compiled file is then saved into a temporary directory. Subsequent calls to the ASP.NET file are processed from the compiled file. The execution of the compiled file on requests is faster than the interpreted environment of Classic ASP.

All in all, ASP.NET is a dramatic improvement over ASP. It has become widely accepted in the development community.

So, What's the Problem?

Based on what you have just read regarding ASP.NET, it may sound really good to you. You may be asking yourself, "Why is there a need for something else? What's the problem?"

The truth is ASP.NET has several issues that need to be addressed:

- ❑ **Round trips** — The server events in ASP.NET require round trips to the server to process these events. These round trips result in all form elements being sent between client and server as well as images and other data files being sent back to the client from the server. While some web browsers will cache images, there can still be significant data transfer.
- ❑ **Speed/network data transfer** — Because of the `VIEWSTATE` hidden form element, the amount of data that is transferred during a postback is relatively large. The more data and controls on the page, the larger the `VIEWSTATE` will be and the more data that must be processed on the server and transmitted back to the client.
- ❑ **Waiting on the result** — When a user clicks on a button or some other visual element that posts data back to the server, the user must wait on a full round trip to complete. This takes time when the processing is done on the server and all of the data, including images and viewstate, are returned to the client. During that time, even if the user attempts to do something with the user interface, that action is not actually processed on the client.

- **User context** — Unless an application is able to properly use the `SMARTNAVIGATION` feature of ASP.NET, the user is redirected to the top of a page by default on a postback. Although there are ways around this issue, this is the default behavior.
- **Processing** — The number of server round trips, the amount of data that is transferred, and the VIEWSTATE element's size result in processing on the server that is not really necessary (Fig. 1-2).

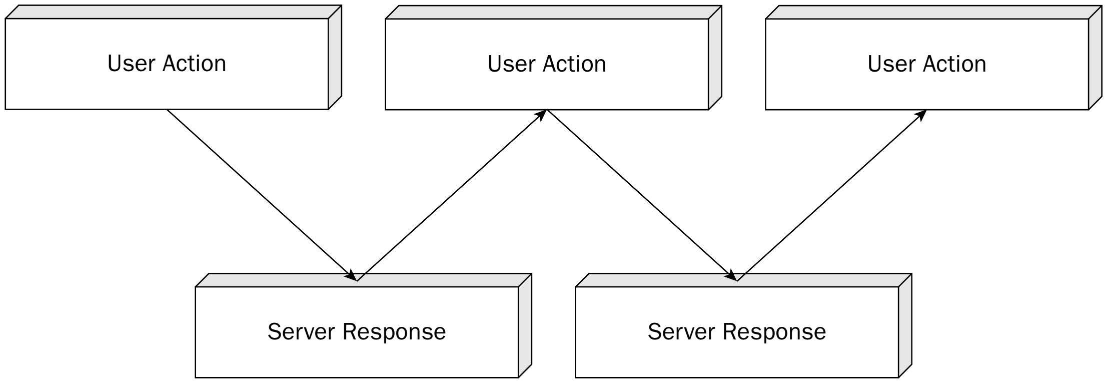

Figure 1-2

Improving the User Experience

Based on these issues, several options present themselves as available for improving the user experience:

- **Java** — Java applets are cross-platform applications. While being used as a cross-platform mechanism to display data and improve the user experience, Java development on the client has not been accepted with open arms into the development community and is primarily used for user interface gee-whiz features as opposed to improving the experience of the user application. (As a side note, Java has been widely accepted for building server-side applications.)
- **XML-based languages** — XML User Interface Language (XUL) and Extensible Application Markup Language (XAML) are two of several languages that can provide an improved user experience. The problem with XUL is that it has been used only in the Mozilla/Firefox line of browsers. XAML is not currently available as a released product. When it is, it will have the problem of being considered a Microsoft-only technology in spite of discussion items like XAML-lite, which has been stated as cross-platform.
- **Flash** — Although Flash has been used and there are cross-platform versions, the product has been used only in the area of graphic UI needs and has not been accepted by the development community as a whole for building line-of-business applications.
- **Ajax** — Ajax is a set of client technologies that provides for asynchronous communication between the user interface and the web server along with fairly easy integration with existing technologies.

Given the amount of recent discussion among developers regarding Ajax, it appears that Ajax has the greatest chance among these technologies of gaining market acceptance.

What Is Ajax?

So into this development environment comes a set of technologies that are collectively referred to as Ajax. If you are an "old guy" developer like us, then Ajax represents a similar concept to the client-server development we mentioned earlier in the chapter. With client-server development, the amount of data transferred is minimized over a terminal application by transferring only the necessary data back and forth. Similarly, with Ajax, only the necessary data is transferred back and forth between the client and the web server. This minimizes the network utilization and processing on the client. Figure 1-3 shows that typically Ajax operates with the assistance of some type of proxy.

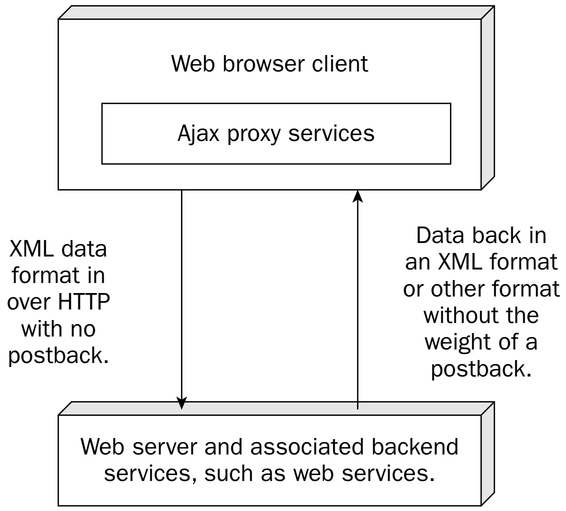
Web browser client
Ajax proxy services
XML data format in over HTTP with no postback.
Data back in an XML format or other format without the weight of a postback.
Web server and associated backend services, such as web services.

Figure 1-3

Advantages of Ajax

The advantages of Ajax over classical web based applications are:

- **Asynchronous** — Ajax allows for the ability to make asynchronous calls to a web server. This allows the client browser to avoid waiting for all data to arrive before allowing the user to act once more.
- **Minimal data transfer** — By not performing a full postback and sending all form data to the server, the network utilization is minimized and quicker operations occur. In sites and locations with restricted pipes for data transfer, this can greatly improve network performance.
- **Limited processing on the server** — With the fact that only the necessary data is sent to the server, the server is not required to process all form elements. By sending only the necessary data, there is limited processing on the server. There is no need to process all form elements, process the viewstate, send images back to the client, and no need to send a full page back to the client.

- ❑ **Responsiveness** — Because Ajax applications are asynchronous on the client, they are perceived to be very responsive.
- ❑ **Context** — With a full postback, the user may lose the context of where they are. The user may be at the bottom of a page, hit the Submit button, and be redirected back to the top of the page. With Ajax there is no full postback. Clicking the Submit button in an application that uses Ajax will allow the user to maintain their location. The user state is maintained, and the user is no longer required to scroll down to the location that he or she was at before clicking Submit.

Figure 1-4 shows how the user interface can still operate while using Ajax. The UI is not locked during the server processing.

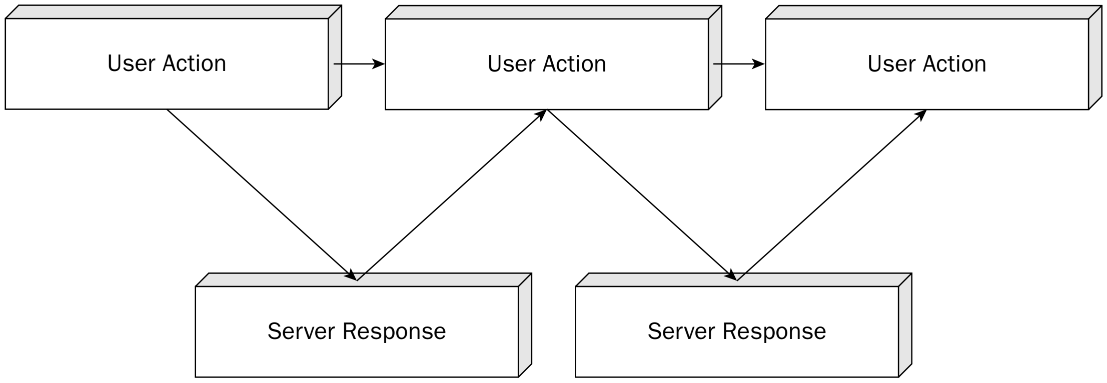

Figure 1-4

History of Ajax

For all of its perceived newness and sexiness, the technologies that make up Ajax are really not new. The ability to communicate back to the server through a hidden frame without posting the main page back to the server has been around for a long time. Communication between client and server has been available — back to the release of Internet Explorer's ability to script ActiveX controls on the client browser and to the MSXML component, both of which date back into the late 1990s. Personally, I saw the first formal usage of client script and MSXML in 2003. The problem with the technology at that time was the need to manually create the necessary client-side JavaScript. In 2003, there was too much code overall that had to be written and too much custom code that had to be written to get this to work. It has been only in the second half of 2005 that client-side libraries and server-side support for ASP.NET have started to make their presence felt and been used significantly.

The mainstream development community has only recently started using the technique. The release of Google's Suggest and Maps are what really opened the eyes of the users to the development technologies. These applications sent a shockwave through the development community.

Technologies That Make Up Ajax

Ajax is a general umbrella term. As mentioned earlier, Ajax itself stands for Asynchronous JavaScript And XML. The term *Ajax* was actually coined by Jesse James Garret of Adaptive Path in an essay that was published in February 2005 (`www.adaptivepath.com/publications/essays/archives/000385.php`) and was quickly accepted by the development community.

Based on this general umbrella term, take a look at the specific items that make up Ajax:

- **XmlHttpRequest** — `XmlHttpRequest` allows the browser to communicate to a back-end server. This object allows the browser to talk to the server without requiring a postback of the entire web page. With Internet Explorer, this capability is provided by the MSXML ActiveX component. With the Mozilla Firefox and other web browsers, this capability is provided by an object literally called `XmlHttpRequest`. The `XmlHttpRequest` object is modeled after the MSXML component. The client-side JavaScript libraries hide the differences between the various browser environments. Sometimes these communications are done through a hidden `FRAME` or `IFRAME`.
- **JavaScript** — JavaScript provides the capabilities to communicate with the back-end server. The JavaScript must be version 1.5 or later. Although JavaScript is not specifically required, it is needed from the standpoint that JavaScript is the only client-side scripting environment supported across the major modern web browsers. There are other client script languages; however, these are not supported across all browsers.
- **DHTML/DOM support** — The browser must support the ability to dynamically update form elements, and the ability to do this in a standard way comes through the support for the Document Object Model (DOM). By supporting the DOM, it becomes easy for developers to write a single piece of code that targets multiple browsers.
- **Data transport with XML or JSON** — Using XML allows for the ability to communicate with the web server in a standard mechanism. There are situations where the JavaScript Object Notation (JSON) is used as the communication notation instead of straight XML.

Running Ajax Applications

Unfortunately, not all web browsers ever produced will support Ajax. To run Ajax, a web browser must:

- Be relatively modern. Ajax style applications are not available in all versions of all web browsers. Whereas Internet Explorer version 6, Firefox version 1.5, and Opera 8.5 provide support for these applications, older versions may be problematic because of their support for different versions of the other requirements.
- Support DOM.
- Utilize JavaScript.
- Support Extensible Markup Language (XML) and Extensible Style Language Transformation (XSLT).
- Possibly have ActiveX enabled on the client. If you are using the Internet Explorer browser while running on Windows, you may have problems if ActiveX is not enabled.

Who's Using Ajax?

Great, now that you have seen that there is this technology called Ajax, are you alone in not having seen or talked about this before? Absolutely not! Ajax has just recently taken off in the second half of 2005 from a mindshare standpoint. As discussions have gone on with counterparts in the development community, many developers are just now looking to what Ajax can do for their applications and ultimately their customers. So, just who is using Ajax publicly?

- ❑ **Google Suggest** — Google Suggest features a dynamic drop-down list box that provides possible items to search on along with the approximate number of search results.
- ❑ **Google Maps** — The ability to grab a map and zoom around without requiring a postback is just amazing. This app/service took the development world by storm when it came out.
- ❑ **Google GMail** — Google GMail is a web-based email system available through Google.
- ❑ **Microsoft Hotmail Kahuna update** — At the time of this writing, the Hotmail upgrade that is referred to as Kahuna is in beta test. As a beta user of the application, I can testify to the improved user interface and responsiveness that this application provides.
- ❑ **Live.com** — The local.live.com service from Microsoft is actively using the Atlas framework, as is nearly the entire Live.com service.
- ❑ **Easy Search component** — The ASP.NET Easy Search Component provides support for searching a single web site similar to the Google Suggest service available through Google.
- ❑ **Other component vendors** — Component vendors such as ComponentArt, Dart, and others are providing controls that provide a rich user experience without forcing a full postback.

In addition to third-party interest, the amount of developer interest is tremendous. For example, one only has to put the word *Ajax* into a blog title to receive an increase in the number of web views. Given the amount of third-party support and the interest of developers, it is only a matter of time before everyone is using it.

Problems Ajax Won't Solve

Ajax is a great technology with a lot of help for typical application problems and a lot of general promise. It will help in areas like network load and user context by sending only the necessary data, creating less network traffic when processing a command. The user is not redirected to the top of a page when a command is sent the server. The problem is that to successfully run Ajax, a user needs to have the transport mechanism and JavaScript support mentioned previously. That sounds like something that any modern browser has, so what's the problem? Well, the problem is that many mobile browsers have support for neither. With many of Ajax solutions, you have limited or no downlevel support, so if you must support a mobile system, you may be limited to writing multiple versions of your application because Ajax may not be the right technology to include.

Summary

Ajax provides developers a foundation to build web-based applications that provide an improved user experience. In this introductory chapter, you have looked at:

- Development from a historical perspective
- Web development methodologies
- Some of the features that Ajax provides, such as improved user responsiveness and decreased load on the web server
- Multiple technologies that can improve the user experience
- Ajax, the problems that it solves, and who is using it

In the next chapter, you are going to examine Dynamic HTML (DHTML). Additional chapters will look at the other technologies that make up Ajax. After that you will examine the Ajax.NET library, Microsoft Atlas, and finally tips and tricks on debugging client-side applications.