Shortly after Programming Atlas was published in September 2006, Microsoft renamed its Ajax framework to “ASP.NET AJAX.” However, this was not the only change. The Atlas functionality was dispersed to one of three packages, or vanished altogether, and a change to the <atlas: tag prefix broke all code listings in the book.

That’s the bad news. The good news is that despite the changes, most Atlas applications are quite easy to migrate.

This Short Cut describes the most important changes that have occurred and provides concise chapter by chapter guidelines for migrating Programming Atlas examples to the final version 1.0 of ASP.NET AJAX. Programming Atlas as a whole will be fully revised and updated for the final release of ASP.NET AJAX in early 2007.
Migrating Atlas applications to ASP.NET AJAX sounds like a rather hard and messy task; all applications have to be changed. However, in most cases, not much effort is required for the migration. There are several reasons for this. First, the most important features of Atlas changed only marginally in ASP.NET AJAX. And second, the most convenient features already exist, not in the ASP.NET AJAX package itself but in the Atlas Control Toolkit, which is now called ASP.NET AJAX Control Toolkit.

This Short Cut mentions important changes for every chapter in Programming Atlas, if applicable. Not all of the functionality used in Programming Atlas survived the switch to ASP.NET AJAX; however, the most important elements of the framework are still there, and continue to work in a very similar fashion.

Of course this Short Cut is not complete and does not cover every aspect that has been changed. It should help you, however, to judge whether it is feasible to port existing Atlas applications to ASP.NET AJAX or not.

Chapter 1: Atlas, Ajax, and ASP.NET

The installation of ASP.NET AJAX is a bit different from the one for the Atlas framework. The ASP.NET team at Microsoft decided to split the package into two components: the fully supported ASP.NET AJAX package, which only contains the framework itself and the most important features; and a Community Technology Preview (CTP) build with additional features that were not considered “important enough” for the core system.

Installation

The home page for both packages is http://ajax.asp.net/. The actual download page (http://ajax.asp.net/default.aspx?tabid=47&subtabid=471) links to both the ASP.NET AJAX (core) build (sometimes called ASP.NET AJAX Extensions) and also to the add-on build (sometimes called ASP.NET AJAX Futures, or ASP.NET AJAX Value-Add CTP). The final version 1.0 of ASP.NET AJAX and the January CTP of the Value-Add package are available at these locations:

http://go.microsoft.com/fwlink/?LinkID=77296
ASP.NET AJAX Extensions

http://go.microsoft.com/fwlink/?LinkID=77294
ASP.NET AJAX Futures

For the sake of simplicity, I’ll refer to the core package as “ASP.NET AJAX” throughout the rest of this document, and the value-add package as “the CTP.” Because the CTP depends on the (core) ASP.NET AJAX package, you need to install ASP.NET AJAX first, and then the CTP; otherwise you will get an error message (see Figure 1).
Figure 1. You have to install the core package first, otherwise you encounter this error message when installing the CTP.

**Warning**

You need to remove older versions of Atlas and ASP.NET AJAX prior to installing the latest ASP.NET AJAX release. Older Atlas applications will continue to work, though, since the Atlas DLL has been copied to each project you created using the Visual Studio template Atlas provided.

After installation, when you open the New Web Site dialog, you’ll see that both packages have added their own web site templates to the list of available templates (see Figure 2).
Figure 2. Each installer sets up one template.

The ASP.NET AJAX installer installs the required files for the AJAX framework just as the Atlas installer did. The core DLL is registered with the GAC (Global Assembly Cache) and therefore does not need to be manually copied into every project. Therefore, the ASP.NET AJAX web site template only consists of a Default.aspx file and a Web.Config file (see Figure 3). The ASP.NET AJAX CTP template additionally puts a file called Microsoft.Web.Preview.dll in the Bin folder.
Tag Prefix

With Atlas, all server-side controls started with `<atlas:`—the tag prefix identified everything that came with the framework. However, since most of the ASP.NET AJAX functionality could end up in ASP.NET in the future, the tag prefix was changed to `<asp:`—that’s right, just like regular ASP.NET web controls. Therefore, the first step when migrating an Atlas application to ASP.NET AJAX is to replace the tag prefixes. The first web control used in Chapter 1 is the `ScriptManager`, which is now called like this:

```xml
<asp:ScriptManager ID="ScriptManager1" runat="server" />  
```

Also, `<atlas:ServiceReference>` needs to be replaced with `<asp:ServiceReference>`. Similar changes are required for all listings, so this fact is not mentioned again in the other chapters.
Web Services

The “Hello World” sample in Chapter 1 of Programming Atlas calls a web method from a local web service and displays the result. Atlas created a JavaScript proxy for the web service and used it to create the SOAP request and to extract the result. It was possible to use a regular ASP.NET 2.0 web service for that task.

This behavior changed with ASP.NET AJAX. Now, “regular” web services need to be altered, since ASP.NET AJAX refuses to create a web service proxy for them. Therefore, the ScriptService attribute must be used (either together with the ScriptService attribute or as its replacement), which is defined within System.Web.Script.Services. Therefore, the WebService.asmx file from Chapter 1 now looks like this:

```csharp
<%@ WebService Language="C#" Class="WebService" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Script.Services;

[ScriptService]
    [WebMethod]
    public string sayHello(string name) {
        return "Hello " + name + ", says the server!";
    }
}
```

**Note**

Prior to the release ASP.NET AJAX RC1, the Microsoft.Web.Script.Services namespace was used instead of System.Web.Script.Services. This namespace change hints that ASP.NET AJAX functionality will go into the ASP.NET core later on.

The Default.aspx file only uses the new tag prefix and does not require any other changes:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>AJAX</title>
    
    <script language="Javascript" type="text/javascript">
        function callService(f) {
            WebService.sayHello(
```
It is also important to note that the support for XML script (which is used quite often throughout the book) has also been moved to the CTP. Finally, the ErrorTemplate property of the ScriptManager control was removed.

**Chapter 2: JavaScript**

No changes due to the ASP.NET AJAX release.

**Chapter 3: Ajax**

No changes due to the ASP.NET AJAX release.
Chapter 4: Controls
The concept of client-side controls was moved to the CTP build of ASP.NET AJAX and no longer resides in the core package.

Client Control Support
So you need to install the CTP and explicitly load the JavaScript library that implements the functionality. To do so, the ScriptManager element must be extended in the following fashion:

```xml
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Scripts>
    <asp:ScriptReference
      Name="PreviewScript.js"
      Assembly="Microsoft.Web.Preview" />
  </Scripts>
</asp:ScriptManager>
```

The syntax may look a bit cumbersome to use every time, but it is mostly a matter of copy and paste.

JavaScript Namespace
The JavaScript namespaces used also changed. Instead of Sys.UI, Sys.Preview.UI must be used. The following updated code for the file ControlMessageBox.aspx shows the two aforementioned changes, which are typical for listings in this chapter:

```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>AJAX</title>
  <script language="JavaScript" type="text/javascript">
    function MessageBoxOKClick() {
      Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window");
    }
    function MessageBoxOKCancelClick() {
    }
    function InputBoxClick() {
      Sys.Preview.UI.Window.inputBox("Using Sys.Preview.UI.Window", "<enter text here>");
    }
  </script>
</head>
<body>
  <form id="form1" runat="server">
```
The most fundamental change, however, is that the $( ) function, which is a shortcut for the JavaScript method document.getElementById(), has been removed. Unfortunately, many other Ajax frameworks have been using the same function name for the same functionality, and have been around much longer than Atlas. (Obviously, other packages “inspired” the Atlas team to use $( ) in the first place.) To avoid any side effects from the duplicate function name, $( ) was renamed to $get( ), which of course must be changed throughout the listings in this chapter. Here is the updated ControlLabel.aspx script:

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>AJAX</title>
    <script language="JavaScript" type="text/javascript">
        window.onload = function() {
            var label = new Sys.Preview.UI.Label($get("Label1"));
            var d = new Date();
            var time = d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds();
            label.set_text(label.get_text() + time);
        }
    </script>
</head>
<body>
<form id="form1" runat="server">
    <span id="Label1">time goes here: </span>
</form>
</body>
</html>
The client-side API remained quite stable, with one notable exception: the `Sys.UI.Select` control was not renamed to `Sys.Preview.UI.Select` (analogous to the other controls), but to `Sys.Preview.UI.Selector`. This code instantiates the control:

```javascript
var selector = new Sys.Preview.UI.Selector($get("Select1"));
```

## Base Methods

The base methods exposed by the client controls changed in a few instances. A typical example is the `ControlCSS.aspx` file. The `get_cssClass()` method ceased to exist and was replaced by the `className` property. The `toggleCssClass()` method was moved into the `Sys.UI.DomElement` namespace (part of the core package). Here is the updated `ControlCSS.aspx` file:

```csharp
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>AJAX</title>
    <style type="text/css">
        .style1 { font-family: Monospace; }
        .style2 { border-style: solid; }
        .style3 { color: #00f; }
    </style>
</head>

<script language="JavaScript" type="text/javascript">
    function pageLoad() {
        window.setInterval(
            function() {
                var label = new Sys.Preview.UI.Label($get("Label1"));
                var rnd = Math.ceil(3 * Math.random());
                Sys.UI.DomElement.toggleCssClass($get("Label1"), "style" + rnd);
                label.set_text($get("Label1").className);
            },
            1000);
        }
    }
</script>

<body>
<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Scripts>
            <asp:ScriptReference
                Name="PreviewScript.js"
                Assembly="Microsoft.Web.Preview" />
        </Scripts>
    </asp:ScriptManager>
</form>
</body>
</html>
```
Control Events
The functionality as presented in *Programming Atlas* has been removed from the framework.

Chapter 5: Data Binding and Validation
The data binding functionality presented in *Programming Atlas* using declarative means still exists in the CTP. Just remember to use `<asp:ScriptReference>` to load the *PreviewScript.js* file.

On the other hand, the validation controls have been removed (up to the RC1 release of ASP.NET AJAX, they were still present, but had been moved into the `System.Web.UI.Compatibility` namespace, hinting that they would eventually vanish altogether); however, those that ship with ASP.NET 2.0 have always been the better choice, anyway. If you still prefer to have AJAX-style validation controls, you may want to have a look at Matt Gibbs’ blog where he provides such controls for download:


Chapter 6: Components and Behaviors
Regarding component behaviors, the syntax changed a bit. Using Atlas, this was a typical XML-script declaration for adding a script behavior to an element:

```xml
  <components>
    <label id="Show1">
      <behaviors>
        <clickBehavior>
          <click>
            <setProperty target="Panel1" property="visible" value="true" />
          </click>
        </clickBehavior>
      </behaviors>
    </label>
  </components>
</page>
```

The name of the behavior and also of the actions (e.g. `<setProperty>`) changed a bit; here is the new version for the CTP:

```xml
  <components>
    <label id="Show1">
      <behaviors>
```

```xml
```
Chapter 7: Animations

With Atlas, all animations were implemented in the AtlasUIGlitz.js file. This hints at the fact that with ASP.NET AJAX, animations have been moved to the CTP release. The following ScriptManager markup loads the animation JavaScript library:

```xml
<asp:ScriptManager ID="ScriptManager1" runat="server">
    <Scripts>
    </Scripts>
</asp:ScriptManager>
```

There are some other changes, as well. However, as a general recommendation, take a look at the Animation framework in the ASP.NET AJAX Control Toolkit, which offers an impressive set of mighty animations and a consistent API.

Chapter 8: Client Script Library

The OOP extensions Atlas brought to the JavaScript language are still there in the ASP.NET AJAX release. However, the EnableScriptComponents attribute of the ScriptManager control has been removed; just load the ScriptManager like this:

```xml
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<% Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.

The StringBuilder API remains identical, as well. However, there is a significant change in the way enumerations are implemented. The prototype pattern is now used (as it is also used internally in the ASP.NET AJAX code), and the getValues() method has been removed. The following listing ports CodeEnum.aspx to ASP.NET AJAX:

```csharp
<%@ Page Language="C#" %>

<asp:ScriptManager ID="ScriptManager1" runat="server" />

This also works in the core distribution of ASP.NET AJAX.
Apart from that, the client library of ASP.NET AJAX extends JavaScript in a variety of ways. Among other things, several helper methods for arrays, strings, and date/time information are implemented. Milan Negovan has published a couple of very handy “cheat sheets” at http://aspnetresources.com/blog/ms_ajax_cheat_sheets_batch2.aspx.

Chapter 9: Using Server Data

The support for ListView and other data controls is still present in ASP.NET AJAX; however, calling the web service throws a server-side exception ("A circular reference was detected..."). This is possibly a bug in the framework; more details on this issue were not available at the time of the final 1.0 release of ASP.NET AJAX.

The data service sample (ListViewDataService.aspx) requires that the ScriptManager control is updated to this markup:

```html
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<div id="output"></div>
</form>
</body>
</html>
```

Also, the web service in the background (ListViewVendorsDataService.asmx) must be altered: the DataService functionality has been moved into the CTP and is now addressed via Microsoft.Web.Preview.Services.DataService:

```csharp
<%@ WebService Language="C#" Class="VendorsDataService" %>
using System;
using System.Web;
```
using System.WebServices;
using System.Data;
using System.Data.SqlClient;
using System.Web.Script.Services;
using System.ComponentModel;

{
    [DataObjectMethod(DataObjectMethodType.Select)]
    public DataTable GetVendors()
    {
        ...
    }
}

The data service is then called, however the server again returns an exception hinting at a possible glitch in the framework.

Chapter 10: Web Services

The ASP.NET AJAX web services support is similar to the support Atlas provided. The most notable change is the new attribute, which has been already mentioned in this document. System.Web.Script.Services.ScriptService must be used to make a web service cooperate with ASP.NET AJAX. For page methods, use the WebMethod attribute as usual. However in order to make these inline methods work, the EnablePageMethods in the ScriptManager element must be set to true:

<atlas:ScriptManager ID="ScriptManager1" runat="server"
    EnablePageMethods="true" />

The web services bridge (.asbx files) was removed in the first pre-release version of ASP.NET AJAX but reappeared in the second beta due to customer feedback.

Chapter 11: Extending Controls

One of the most appealing aspects of working with an Ajax framework is to reuse functionality that ships with the code. Chapter 11 covers several of those Ajax controls.

Drag and Drop

The DragOverlayExtender control is still part of ASP.NET AJAX—the CTP version. You do not need extra markup within the ASP.NET page to add support for the extender. However, the syntax changed a bit. Here is the old version:

<atlas:DragOverlayExtender runat="server">
    <atlas:DragOverlayProperties Enabled="true" TargetControlID="DragPanel" />
</atlas:DragOverlayExtender>

The pattern that put all properties of a control in an XXXProperties subelement is not used any longer; instead, properties are now direct attributes of the actual
control. Therefore, the previous code (taken from *DragDrop.aspx*) must be changed to this syntax:

```
<asp:DragOverlayExtender ID="DragOverlayExtender1" runat="server"
    Enabled="true" TargetControlID="DragPanel" />
```

**Autocomplete**

The `AutoCompleteExtender` control was first moved to the CTP and finally found a new home in the ASP.NET AJAX Control Toolkit. Also, all properties that were once set using `AutoCompleteExtenderProperties` are now direct control attributes:

```
<ajaxToolkit:AutoCompleteExtender ID="AutoCompleteExtender1" runat="server"
    ServicePath="Vendors.asmx" ServiceMethod="GetVendors"
    TargetControlID="vendor" />
```

And of course, do not forget to use `ScriptService` attribute in the `Vendors.asmx` service.

**UpdatePanel**

The `UpdatePanel` control proved to be so popular that this extender is part of the ASP.NET AJAX core package. Actually, a number of applications that use ASP.NET AJAX use the `UpdatePanel` control almost exclusively.

In previous Atlas versions, the `UpdatePanel` functionality was only present if the `EnablePartialRendering` attribute of the `ScriptManager` control had to be set to `true`. This changed in ASP.NET AJAX: partial rendering is now enabled by default, making the `UpdatePanel` control even easier to use.

One thing that might have to be changed when migrating an Atlas app to ASP.NET AJAX is your use of triggers. Previously, triggers (`<atlas:ControlEventTrigger>`) worked asynchronously, but with ASP.NET AJAX, developers have the choice of asynchronous triggers (`<asp:AsyncPostBackTrigger>`) or synchronous ones (`<asp:PostBackTrigger>`). Here is an updated version of the script `UpdatePanelTimer.aspx`:

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        CurrentTime.Text = DateTime.Now.ToString();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
```
<head id="Head1" runat="server">
<title>AJAX</title>
</head>
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:TimerControl ID="FiveSeconds" Interval="5000" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Label ID="CurrentTime" runat="server" />
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="FiveSeconds" EventName="Tick" />
    </Triggers>
</asp:UpdatePanel>
</form>
</body>
</html>

Also, an important attribute of the UpdatePanel control has changed, from Mode to UpdateMode. This was used in the UpdatePanelTimerCode.aspx sample, which must be changed to this:

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        CurrentTime.Text = DateTime.Now.ToLongTimeString();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>AJAX</title>
</head>
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:TimerControl ID="FiveSeconds" Interval="5000" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Label ID="CurrentTime" runat="server" />
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="FiveSeconds" EventName="Tick" />
    </Triggers>
</asp:UpdatePanel>
</form>
</body>
</html>
Chapter 12: Virtual Earth
The functionality as presented in *Programming Atlas* has been removed from the framework.

Chapter 13: Web Parts and Gadgets
The gadgets functionality as presented in *Programming Atlas* has been removed from the framework; web parts, however, continue to exist (under the new prefix).

Chapter 14: Atlas Control Toolkit
The Atlas Control Toolkit is now called ASP.NET AJAX Control Toolkit. It is updated quite often, and—amazingly—is always in sync with ASP.NET AJAX. The tag prefix registered when using the Control Toolkit Visual Studio template is `ajaxToolkit`. The whole project resides at [http://ajax.asp.net/default.aspx?tabid=47&subtabid=477](http://ajax.asp.net/default.aspx?tabid=47&subtabid=477) and is hosted at CodePlex. It now features over 30 controls, and more are being added.

Creating a custom control using the Control Toolkit infrastructure has changed significantly. This is the (a bit simplified) JavaScript code from the `FilteredTextBox` control—the control that is based on the `TextBoxMask` sample control from *Programming Atlas*!

```javascript
// (c) Copyright Microsoft Corporation.
// This source is subject to the Microsoft Permissive License.
// All other rights reserved.

Type.registerNamespace('AjaxControlToolkit);

AjaxControlToolkit.FilteredTextBoxBehavior = function(element) {
    AjaxControlToolkit.FilteredTextBoxBehavior.initializeBase(this, [element]);

    this._keypressHandler = null;
    this._changeHandler = null;

    this._filterType = AjaxControlToolkit.FilterTypes.Custom;
    this._validChars = null;

    this.charTypes = new Object();

    this.charTypes['LowercaseLetters'] = 'abcdefghijklmnopqrstuvwxyz';
    this.charTypes['UppercaseLetters'] = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    this.charTypes['Numbers'] = '0123456789';

    initialize : function() {
        AjaxControlToolkit.FilteredTextBoxBehavior.callBaseMethod(this, 'initialize');
    }
}
```

AjaxControlToolkit.FilteredTextBoxBehavior.prototype = {

    initialize : function() {
        AjaxControlToolkit.FilteredTextBoxBehavior.callBaseMethod(this, 'initialize');
    }
}````
var element = this.get_element();

this._keypressHandler = Function.createDelegate(this, this._onkeypress);
$addHandler(element, 'keypress', this._keypressHandler);

this._changeHandler = Function.createDelegate(this, this._onchange);
$addHandler(element, 'change', this._changeHandler);
},

dispose : function() {
  var element = this.get_element();

  $removeHandler(element, 'keypress', this._keypressHandler);
  this._keypressHandler = null;

  $removeHandler(element, 'change', this._changeHandler);
  this._changeHandler = null;

  AjaxControlToolkit.FilteredTextBoxBehavior.callBaseMethod(this, 'dispose');
},

_getValidChars : function() {
  if (this._validChars) return this._validChars;

  this._validChars = "";

  for (type in this.charTypes) {
    var filterType = AjaxControlToolkit.FilterTypes.toString(this._filterType);

    if (filterType.indexOf(type) != -1) {
      this._validChars += this.charTypes[type];
    }
  }

  return this._validChars;
},

get_ValidChars : function() {
  return this.charTypes['Custom'];
},

set_ValidChars : function(value) {
  if (this._validChars != null || this.charTypes['Custom'] != value) {
    this.charTypes['Custom'] = value;
    this._validChars = null;
    this.raisePropertyChanged('ValidChars');
  }
},

get_FilterType : function() {
  return this._filterType;
},

set_FilterType : function(value) {
  if (this._validChars != null || this._filterType != value) {
    this._filterType = value;
    this._validChars = null;
  }
this.raisePropertyChanged('FilterType');
},

_{onkeypress : function(evt) {
  var scanCode;

  if (evt.rawEvent.keyIdentifier) {
    // Safari
    // Note (Garbin): used the underlying rawEvent insted of the DomEvent instance.
    if (evt.rawEvent.ctrlKey || evt.rawEvent.altKey || evt.rawEvent.metaKey) {
      return;
    }
    if (evt.rawEvent.keyIdentifier.substring(0,2) != "U") {
      return;
    }
    scanCode = evt.rawEvent.charCode;
  }
  else {
    scanCode = evt.charCode;
  }

  if (scanCode && scanCode >= 0x20 /* space */) {
    var c = String.fromCharCode(scanCode);
    if(!this._processKey(c)) {
      evt.preventDefault();
    }
  }
},

_processKey : function(key) {
  var filter = this._getValidChars();
  return (!filter || filter.length == 0 || filter.indexOf(key) != -1);
},

_onchange : function() {
  var text = this.get_element().value;
  var i = 0;
  var chars = this._getValidChars()
  while (i < text.length) {
    if (chars.indexOf(text.substring(i, i+1)) == -1) {
      text = text.substring(0, i) + text.substring(i+1, text.length);
    } else {
      i++;
    }
  }

  this.get_element().value = text;
The walkthrough on the official web site [http://ajax.asp.net/ajaxtoolkit/Walkthrough/CreatingNewExtender.aspx](http://ajax.asp.net/ajaxtoolkit/Walkthrough/CreatingNewExtender.aspx) has also been updated to the new API.

**Chapter 15: Using Atlas with Other Server Technologies**

The internal syntax of the JavaScript web service proxy may have changed a bit, but the general approach presented in this chapter remains the same. The server-side portions of Atlas/ASP.NET AJAX need to be emulated; then, the client-side functionality continues to work.

Starting with ASP.NET AJAX 1.0, the client-side JavaScript library portion of the framework can be downloaded separately under the name *Microsoft AJAX Library* ([http://ajax.asp.net/downloads/library/default.aspx?tabid=47](http://ajax.asp.net/downloads/library/default.aspx?tabid=47)). Since this version only consists of JavaScript code, it can also be used with other server technologies like PHP—or no server technology at all! CodePlex features a project that facilitates using the Microsoft AJAX Library with PHP. The PHP code there is based on the (somewhat simplified) functionality presented in this chapter. You can find out more at [http://codeplex.com/phpmsajax](http://codeplex.com/phpmsajax).

**Chapter 16: Other Ajax Tools**

No changes due to the ASP.NET AJAX release.

**Appendix**

As usual, the ASP.NET AJAX homepage ([http://ajax.asp.net/](http://ajax.asp.net/)) is the best place to look for up-to-date information. For the final release of ASP.NET AJAX, the following information provides you with more details about the new release and further migration practices:


Migration from Atlas to ASP.NET AJAX 1.0
Migration from ASP.NET AJAX RC1 to ASP.NET AJAX 1.0

http://ajax.asp.net/files/Migration_Guide_RC_to_RTM.aspx

Detailed document describing changes between Atlas and ASP.NET AJAX 1.0

http://ajax.asp.net/files/AspNet_AJAX_CTP_to_RC_Whitepaper.aspx

Finally, when Scott Guthrie announced the RC1 release in mid-December 2006 in his weblog at http://weblogs.asp.net/scottgu/archive/2006/12/15/asp-net-ajax-1-0-release-candidate-now-available.aspx, he included a great tip: to get IntelliSense to immediately work with the latest version of ASP.NET AJAX, delete the schema file cache that has probably been filled by previous Atlas versions. You’ll find it in the C:\Documents and Settings\<Username>\Application Data\Microsoft\Visual Studio\8.0\ReflectedSchemas folder; Vista users have to look for it in C:\Users\<Username>\AppData\Roaming\Microsoft\Visual Studio\8.0\ReflectedSchemas.

The release announcement for the final release, also coming from Scott Guthrie, can be found at http://weblogs.asp.net/scottgu/archive/2007/01/23/asp-net-ajax-1-0-released.aspx.