Microsoft ASP.NET Ajax Road Map

**Microsoft Corporation Documentation License Agreement**

THIS IS A LEGAL AGREEMENT BETWEEN MICROSOFT CORPORATION ("MICROSOFT") AND THE RECIPIENT OF THESE MATERIALS, WHETHER AN INDIVIDUAL OR AN ENTITY ("YOU"). BY ACCESSING, USING OR PROVIDING FEEDBACK ON THE ATTACHED MATERIALS, YOU AGREE TO THESE TERMS.  
  
1. Microsoft grants You the right to reproduce and review this documentation (a) as a reference to assist You in planning and designing Your product, service or technology (“Product”) to interface with a Microsoft Product, as described in these Materials; and (b) to provide feedback on these Materials to Microsoft. All rights not expressly granted herein are reserved by Microsoft; this agreement does not give You rights under any Microsoft patents.

2. This is a preliminary document and may be changed substantially prior to final commercial release of the Microsoft Product described herein. THIS DOCUMENTATION IS MADE AVAILABLE AS IS AND MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION CONTAINED IN IT.

3. You have no obligation to give Microsoft any suggestions, comments or other feedback ("Feedback") relating to these Materials. However, any Feedback You voluntarily provide may be used in Microsoft Products and related specifications or other documentation (collectively, "Microsoft Offerings") which in turn may be relied upon by other third parties to develop their own Products. Accordingly, if You do give Microsoft Feedback on any version of these Materials or the Microsoft Offerings to which they apply, You agree: (a) Microsoft may freely use, reproduce, license, distribute, and otherwise commercialize Your Feedback in any Microsoft Offering; (b) You also grant third parties, without charge, only those patent rights necessary to enable other Products to use or interface with any specific parts of a Microsoft Product that incorporate Your Feedback; and (c) You will not give Microsoft any Feedback (i) that You have reason to believe is subject to any patent, copyright or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any Microsoft Offering incorporating or derived from such Feedback, or other Microsoft intellectual property, to be licensed to or otherwise shared with any third party.

In this document we will describe some of the proposed features that we are considering investing in future releases of ASP.NET AJAX, Visual Web Developer, and the ASP.NET AJAX Control Toolkit.

We really appreciate your feedback, so this document is intended as much for you to provide an input to our direction as well as to give some indication of what the teams are investing in.

Ajax frameworks have matured. Developers demand further higher-level helpers, components and functionality, not just a base library with class, interface and other OOP definitions. We want to make ASP.NET Ajax the first-class choice for all Web developers by enabling a lot more down to the metal, pure client-side scenarios, including on mobile devices, and integrating the framework with the new server-side MVC framework. This should provide developers who live on the bleeding edge of technology with the foundation, extensibility points and total control that they need.

We also want to dramatically simplify client-centric Ajax development and UI experiences, bringing easy to use solutions to some of the hardest problems of client-side developments. This includes new and more intuitive templates, data-binding and declarative syntax, as well as client-side data, new controls and even better integration with the server.

In terms of the tools experience, we will further improve JavaScript IntelliSense, JavaScript debugging and add support for navigation and building JavaScript libraries.

Before we delve into this, remember that the Microsoft ASP.NET Ajax framework can be used entirely as a client-side technology leveraging specific ASP.NET services or data, or even without an ASP.NET server, by utilizing the Microsoft ASP.NET Ajax libraries directly.

Contents

[Ship Roadmap 4](#_Toc201993651)

[Framework Improvements and Enhancements 4](#_Toc201993652)

[Make ASP.NET Ajax the first-class choice for all Web 2.0 developers 4](#_Toc201993653)

[DOM APIs 4](#_Toc201993654)

[Enable great mobile Ajax applications 5](#_Toc201993655)

[Make client-centric Ajax development and UI experiences easy 6](#_Toc201993656)

[Client Data and UI Templates 6](#_Toc201993657)

[Client Data Sources 8](#_Toc201993658)

[Client Data and Server Data 8](#_Toc201993659)

[Accessibility 9](#_Toc201993660)

[Animation 9](#_Toc201993661)

[Drag & Drop 9](#_Toc201993662)

[Client-side Controls and Behavior 10](#_Toc201993663)

[Ajax and MVC 10](#_Toc201993664)

[Interoperability 11](#_Toc201993665)

[Provide a great tools experience for Ajax development 11](#_Toc201993666)

[Reduced update lag for JavaScript IntelliSense 11](#_Toc201993667)

[JavaScript IntelliSense support for a broader range of coding styles 11](#_Toc201993668)

[Recognizing XML Metadata Comments in the Active Document 11](#_Toc201993669)

[Recognizing OpenAjax-compliant XML Metadata Files 11](#_Toc201993670)

[JavaScript Navigation Tools 11](#_Toc201993671)

[JavaScript Build Tools 12](#_Toc201993672)

[Other Potentials 12](#_Toc201993673)

[High-level Feature Proposal Breakdown 12](#_Toc201993674)

[Conclusion 12](#_Toc201993675)

# Ship Roadmap

Our first generation of AJAX offerings were shipped in several vehicles: a standalone ASP.NET AJAX V1 release (which was then integrated into .NET Framework 3.5), a client-only Microsoft AJAX Library, a set of future features in the ASP.NET Futures, a toolset in Visual Studio 2008, and a set of controls and components in the ASP.NET AJAX Control Toolkit.

Going forward, we are looking at simplifying this model into two major categories, to reduce confusion and create an approach consistent with our other offerings. We’d like your feedback on this approach.

1. Fully supported framework and tools features will be released together, in full or update versions of our framework and tools releases. These will have CTPs, Betas, etc.
2. We will offer our AJAX components on CodePlex, which will provide a single place to obtain updates to source code components, including the ASP.NET AJAX Control Toolkit, as well as features not yet part of the framework.

We may have different landing pages, preview vehicles, etc. from time to time, but the two vehicles above will provide a definitive way to get our ASP.NET AJAX offerings.

# Framework Improvements and Enhancements

These proposals are not defined in any priority order.

## Make ASP.NET Ajax the first-class choice for all Web 2.0 developers

### DOM APIs

**History**: Microsoft ASP.NET Ajax 3.5 has DOM features that enable users to easily find elements ($get), get their coordinates (Sys.UI.DomElement.getBounds and getLocation), add and remove event handlers (Sys.UI.DomEvent) and manipulate their CSS classes (Sys.UI.DomElement.addCssClass, containsCssClass, toggleCssClass and removeCssClass) using a uniform, cross-browser API. We had also been investigating further CSS selection of elements in the Microsoft ASP.NET “Futures” release using the CSS1 selector syntax.

Developers typically want a rich API set to allow for all the UI generation and handling that rich (Ajax) Internet Applications now demand. In v-next, we are considering greatly enhancing DOM APIs and manipulation by:

* CSS 2.1 DOM selection with support for native querySelector APIs. This means many APIs related to parents, children, attributes, Css classes and styles in terms of selection, generation and manipulation. The following example will find all input elements of type text and whose value contains “foo”.

$query(“input[type=text][value~=foo]”);

* Chainable DOM generation and manipulation APIs. This means the ability to create user code that utilizes selection and manipulation in a single statement for example, reducing code size. The following example will find all TEXTAREA elements with class “rich”, attach an event to each one of them that outputs a message to the trace every time the TEXTAREA is focused, set their width to the width of the document minus 10 pixels and attaches rich text behaviors to them.

$query(“textarea.rich”)  
 .addHandler(“focus”, function(e) {  
 Sys.Debug.trace(“focused into “ + (e.eventTarget.id || “?”));  
 })  
 .setStyle(“width”, function() {  
 return (document.body.clientWidth – 10) + “px”;  
 })  
 .create(Contoso.UI.RichTextBehavior, {  
 showToolbar: true,  
 fonts: [“Arial”, “Times”, “Courier”]  
 });

* CSS selector-based event sink so that it becomes possible to set-up an event for all elements that satisfy a given selector at the time when the sink is created and later. This means that events could be generated at the window or component level, without the need to perform clean-up and re-generation when HTML elements are destroyed and re-created and manage their own event hook-ups. The following example shows setting up an event sink that will result in the title attribute of all spans getting dumped to the debug trace. This starts working at the moment the call is made and will continue working even for new spans that get created after that.

$listen(“hover”, “span[title]”, function(e) {  
 Sys.Debug.trace(this.title);  
});

### Enable great mobile Ajax applications

Mobile access to Web applications is becoming more and more important as phones gain capabilities and approach desktop capabilities. We want to look at enabling the new generation of mobile browsers to run ASP.NET Ajax applications.

## Make client-centric Ajax development and UI experiences easy

### Client Data and UI Templates

**History**: If you have been following Microsoft ASP.NET Ajax from its “Atlas” days, you may recall that we introduced a declarative model for JavaScript objects called “xml-script”. This had had many elegant features and enabled live data bindings and rich UI templates. It was also very complex to understand and had performance issues that limited its success. It also meant separation of the XHTML markup defining the UI and templates from the declarative markup that performed such templates and binding. This means potentials for the two to be out-of sync and also introduced yet another ‘concept’ in an unfamiliar way.

In v-next we want to reintroduce client-side data and templates. This means the ability to easily declare a re-useable UI template that is bound to repeated data of some form. In particular, we want to make it as easy as possible for a client-side application to consume data from Web Services or ADO.NET Data Services. Displaying this data should be simple and flexible. We want to make the most common and simple scenario of one-time, one-way binding expressible in the simplest possible way. This should also enable a richer live, bi-directional bindings where needed.

Our initial thoughts mean the ability to define templates in markup directly in the page, and apply bindings on the markup itself. We will do this with the flexibility that JavaScript brings as an expression language and as a way to enrich the contents beyond plain HTML. Performance is of course a high priority, so our goals will ensure that this model can handle rich UI generation in a highly efficient manner.

<div id=”repeater1”></div>

<div id=”template1” class=”sys-template”>  
 <h2><a href=”{{ ‘products/’ + id }}”>{{name}}</a></h2>  
 <p>{{description}}</p>  
</div>  
  
<script type=”text/javascript”>  
 Sys.Application.add\_initialize(function() {  
 $create(Sys.UI.DataView, {  
 template: $get(“template1”),  
 data: myData  
 }, {}, {}, $get(“repeater1”));  
}  
</script>

The above example repeats the html described in the template element over a data set and replaces parts of the rendering with data. Notice how arbitrary JavaScript expressions can be used in templates, e.g.({{ ‘products/’ + id }}).

<body xmlns:sys=”javascript:Sys” xmlns:dv=”javascript:Sys.UI.DataView”>  
…  
<div id=”repeater1” sys:attach=”dv” dv:data=”{{someArray}}” class=”sys-template”>  
 <h2><a href=”{{ ‘products/’ + id }}”>{{name}}</a></h2>  
 <p>{{description}}</p>  
</div>

The above example is equivalent to the previous example, but uses a declarative approach to specifying the template and the data.

<body xmlns:sys=”javascript:Sys” xmlns:dv=”javascript:Sys.UI.DataView”>  
…  
<div id=”tripList” sys:attach=”dv” dv:data=”{{myData}}”   
 dv:template=”{{$get(‘template2’)}}”></div>  
  
<div id=”template2” class=”sys-template”  
 xmlns:ac=”javascript:Sys.UI.AutoComplete”  
 xmlns:wm=”javascript:Sys.UI.Watermark”  
 xmlns:dp=”javascript:Sys.UI.DatePicker”  
>

<input type=”text” sys:id=”{{ ‘airport’ + $index }}”  
 sys:attach=”ac,wm”  
 ac:serviceUrl=”airportList.asmx”  
 ac:minimumPrefixLength=”{{1}}”  
 wm:text=”Type the name of an airport”  
 value=”{Binding airport, mode=twoWay}” />  
 <input type=”text” sys:id=”{{ ‘flight’ + $index }}”  
 value=”{Binding flight, mode=twoWay}” />  
 <input type=”text” sys:id=”{{ ‘date’ + $index }}”  
 sys:attach=”dp”  
 dp:lowerBound=”{{ new Date(1970, 4, 21) }}”  
 dp:upperBound=”{{ new Date(2050, 1, 1) }}”  
 value=”{Binding date, mode=twoWay}” />

</div>

The above example is a more complex template. Here, the default one-way, one-time data bindings we had before are replaced with bidirectional, live bindings that are declared with a syntax that is reminiscent of WPF bindings. In addition to the generation of HTML, the template also includes the declarative creation of JavaScript behaviors attached to the HTML elements.

Mappings between attribute prefixes that enable XHTML-compliant declaration and JavaScript types are declared on body or on the template’s outer tag (xmlns attributes). These prefixes are then used to declare which behaviors get attached to an element (sys:attach=”ac,wm”) and to set simple property values on them (wm:text=”Type the name of an airport”).

This chapter is added to this document only to make it slightly different from the original document.

It is also possible to create behaviors inside templates by writing plain JavaScript code inside <!--\* \*--> blocks, which enables complete flexibility. The template engine exposes the latest created HTML element as $element and the index of the current data item as $index.

<body xmlns:sys=”javascript:Sys” xmlns:dv=”javascript:Sys.UI.DataView”>  
…  
<div id=”productList” sys:attach=”dv” dv:data=”{{myProducts}}”   
 dv:template=”{{$get(‘template3’)}}”></div>  
<div id=”template3” class=”sys-template”>  
 <h2>{{name}}</h2>  
 <!--\* if (description) { \*-->  
 <p>{{description}}</p>  
 <!--\* } \*-->  
 <!--\* for (var i = 0; i < features.length; i++) { \*-->  
 <span>{{ features[i].name }}</span>  
 <!--\* $create(Contoso.Tooltip, {  
 text: features[i].description  
 }, {}, {}, $element);  
 \*-->  
 <!--\* } \*-->  
</div>

### Client Data Sources

The client UI templates will support both one-way/one time binding to a simple JavaScript Array, and ‘live binding’ where changes in the data automatically propagate to update the rendered UI (as a result of either property-change events or collection-change events) and *vice-versa*.

To support these scenarios, we expect to implement a client *DataSource* class, which can:

* specify a source of data, such as an ADO.NET Data Service.
* request data from the source.
* cache data.
* save changes back to source.
* Expose methods such as insertRow.
* provide collection-change events on cached data.

The client DataView control can bind to a client DataSource, through a live binding.

In the following example the declarative syntax is used to bind the DataView (and its inline template) to a DataSource which uses an ADO.NET Data Service to request or save data:

<body xmlns:sys=”javascript:Sys”   
 xmlns:dv=”javascript:Sys.UI.DataView”  
 xmlns:ds="javascript:Sys.Data.DataSource">  
…  
<span sys:attach="ds"   
 ds:sys-key="customersDataSource"   
 ds:provider="{{ myNorthwindDataService }}"   
 ds:select="Customers" >  
</span>

<div sys:attach="dv" dv:data="{{ customersDataSource }}" class="sys-inline-template">  
 <h4>{{ $index+1 }}. {{ CompanyName }}</h4>  
 <b>{{ Address }} {{ City }} {{ PostalCode }}</b> <br /><br />  
 <i>Contact:</i> {{ ContactName }} <br />  
</div

### Client Data and Server Data

In addition to Data Services, we propose to provide more convenient integration between client data and server data, for page developers.

Among the integration features we are considering:

A **ClientDataSource** server control – which is rendered in the browser as a client DataSource control, but which can be initialized from server data or code-behind. The server control can be updated with current data from the client DataSource, during postback (as postback data), and possibly through the save method on the client DataSource (using lightweight XmlHttpRequest calls). The data can be accessed on the server via the server control, from code-behind, or through server data binding.

A **ClientDataSourceExtender** control – used to extend the behavior of a normal server DataSource control in a similar way to the above ClientDataSource, so that it is exposed as a client DataSource control in the browser.

A **ClientDataView** server control, which can be data-bound directly to a server DataSource control, but is rendered into the browser as a client DataView using client templating and data-binding. The ClientDataView will ensure that the server DataSource control is rendered as a client DataSource with source pointing back to the server DataSource control.

### Accessibility

**History**: We had previously investigated accessibility back in the Microsoft ASP.NET Ajax 1.0 days, but at that time browsers had little support for WAI-ARIA.

Given the maturing of existing and new accessibility standards we want to provide clear guidance and well-defined patterns for both component and application developers to achieve accessible compatible Ajax applications. This means that in any script components that we create we will implement accessibility patterns in all components that we develop and provide these along with clear guidance documentation for third party component and application vendors.

This translates into APIs that we develop in the core framework to make accessibility easy to implement. We will provide support and guidance for keyboard navigation and focus management, screen readers, WAI-ARIA roles, high contrast and alternate contents.

A stretch goal, but an important goal, would be to also provide ways to verify the accessibility of a component or application.

### Animation

To provide rich behavior in the framework, we want to provide a consistent core animation framework much like that we have currently in the ASP.NET “Futures”. This will include simple animations such as fade or slide, as well as ways to combine them, allowing developers to create obvious transition effects between UI changes for example.

We think that the core framework will be highly extensible, allowing 3rd parties or application developers to easily extend and enrich.

$query(“.sprite”).animate([  
 new Sys.Animation.FadeIn(300),  
 {  
 “style.backgroundColor”: “#ff0000”,  
 “style.fontSize”: “2em”),  
 duration: 500  
 },  
 new Sys.Animation.FadeOut(300)  
 ])  
);

This sample selects all elements with class “sprite” and animate them with first a fade-in effect that will last 300 milliseconds, then will transition their background colors to red while taking their font size to 2em. Finally, a 300 millisecond fade-out effect is applied.

### Drag & Drop

We want to provide a flexible drag and drop framework and integrate it with components such as DataView.

### Client-side Controls and Behavior

**History**: Currently the Microsoft ASP.NET Ajax framework does not ship any higher-level ‘components’ or controls in the server or client. There are some server-side controls that provide Ajax-like behaviors for ASP.NET server applications, UpdatePanel and UpdateProgress. The higher-level UI components have been provided by third parties and by the [Ajax Control Toolkit](http://asp.net/ajax/ajaxcontroltoolkit/).

Many developers see the Ajax Control Toolkit as an integral part of the Microsoft Ajax stack, and so do we. We want to affirm our commitment to the Toolkit by dedicating more resources to creating new controls, fixing bugs and improving the existing controls.

The Ajax Control Toolkit will continue to be a community project but in our v-next release we are considering developing new simple, core client-side components together with server-side controls and extenders. We have not yet determined the set of components that we might offer, but certainly in terms of the templates, data-binding and UI generation mentioned earlier, we are considering specific base data-bound controls such as Grid, DataView, as well as Validators (including an asynchronous custom validator that enables client-side code to validate fields against a custom server rule).

We would also consider Layout controls to support UI generation and other navigation-style controls such as TreeView.

Which controls get implemented first will highly depend on community feedback. We welcome suggestions and encourage you to communicate your preferences about which controls would be the most useful to you as well as what features you would expect on each of them. We identified the following controls as possible candidates:

* **Grid**: a client-side, templated grid control with client-side sorting, pagination and inline editing.
* **TreeView**: a client-side, templated and web-service aware tree with a full client-side object model.
* **Asynchronous validator:** enables asynchronous validation using custom server-side code.
* **Upload**: an upload control compatible that works outside of full postbacks, possibly with progress monitoring.
* **Layout**: a control that enables the easy construction of simple and dynamic layouts with user-configurable splitters.
* **User control**: an easy pattern to build reusable pieces of UI, analogous to server-side user controls.
* **Rich text editor**: a simple, cross-browser and extensible rich text editor.
* **Chart**: simple and extensible charting, possibly based on SilverLight.
* **Progress bar**: a skinnable control to convey progress information on long-running processes.
* **Color picker**: a control to pick colors.

### Ajax and MVC

With the new complementary model for Web application development from ASP.NET, MVC, we want to ensure that the full suite of Microsoft Ajax functionality can be utilized through this paradigm, including the Ajax Control Toolkit. MVC developers want complete control over the rendering of the page and are typically more willing to write JavaScript code. In that spirit, we want to enable:

* **Partial rendering scenarios**: We need to ensure that developers can create updateable areas of a Web page, or partial views, that do not require a full-page refresh. We also need to enable developers to easily create links to actions that will update the partial views. To do this, we need to build the client-side infrastructure that updates the DOM, downloads new scripts that may have been added during the partial update, and create the server-side object model that controls it. We believe that this can be accomplished using UserControls as partial views.
* **Controls and behaviors:** We want to ensure that you can incorporate the RIA Microsoft ASP.NET Ajax behaviors and controls as well as register scripts through simple rendering helpers exposed off the “Ajax” property in the ViewPage.

### Interoperability

Microsoft has recently set its commitment to interoperability as one of its top priorities. As part of that effort, we have been actively participating in the activities of the OpenAjax Alliance and in particular the Interop group that produces the OpenAjax hub.

The OpenAjax hub enables disparate Ajax libraries and components to communicate with each other, such as Microsoft ASP.NET Ajax and Prototype or Dojo. While Microsoft ASP.NET Ajax is compatible with the OpenAjax hub today, we want to make that easier for the application developer by providing a simple API to expose any Microsoft ASP.NET Ajax event as an OpenAjax publisher. The library will also properly register itself to the hub if it is present.

We think that this can be built in a way that the current developer on Microsoft ASP.NET Ajax is unaffected, and that it is really an application developer’s decision to opt-in for. This allows the functionality to be ‘paid-for’ rather than bloating the framework even if it is unused.

## Provide a great tools experience for Ajax development

Here is a list of ideas under consideration for Visual Studio Ajax Editing Support. Please note that a few proposals may be implemented in the next major release, some may be realized in subsequent release, and others may not be realized at all. Of source, feedback is welcome to help us prioritize.

### Reduced update lag for JavaScript IntelliSense

For larger script libraries, JavaScript IntelliSense may take longer than usual to update. We are investigating methods to reduce this wait time.

### JavaScript IntelliSense support for a broader range of coding styles

Much work has been done in Visual Studio 2008 SP1 to ensure IntelliSense compatibility with popular JavaScript libraries. In general, we would like to achieve IntelliSense compatibility with nearly any type of coding style.

### Recognizing XML Metadata Comments in the Active Document

Currently, in order to see additional metadata in IntelliSense (type-information, summary text) for a member, that member must be defined in an externally referenced file. We would like to remove this restriction and allow this metadata to be available regardless of where it is defined.

### Recognizing OpenAjax-compliant XML Metadata Files

The OpenAjax Alliance is working to increase the interoperability of Ajax libraries by creating an open standard for the description of these libraries. Our work here in the editor would be to natively read this data and merge it into our existing tooltips and IntelliSense.

Another chapter just for fun…

### JavaScript Navigation Tools

We’ve heard many requests in the category of “navigation aids for large files”. The current thinking is to provide a document outline tree view that allows someone to get a quick overview of a large file and also to jump to any line via the tree. The possibilities for the tree representation range from purely lexical to something more akin to the class view. For example, you would see a function and its prototype grouped together regardless of where they were defined.

### JavaScript Build Tools

We will be investigating methods to integrate common JavaScript related tasks into the build process. This eliminates the need to manually invoke a checklist of tools and the possibility for forgetting one. Some possible modules include:

* **Aggregate** – In release build mode, merge multiple script files into one. This allows for the development-time file structure to be optimized for componentization and readability.
* **Minify** – In release build mode, compact your script to optimize for download speed.
* **Static Analysis** – Run basic checks against your file such as comparing documented function signatures against the actual ones to identify subtle bugs.

# Other Potentials

We have many other ideas that we might simply produce and release as sample code, or into our “Futures” buckets and so does not become part of the supported framework for now. However, we’d like your input for these.

Typically, we are considering an application profiling tool that allows you to inspect a particular page and understand its library script requirements. The tool will generate a list of scripts in their relevant order that will allow you to easily take advantage of the .NET FX 3.5 SP1 script combining feature on the ScriptManager control. This means you can further optimize the performance of your application.

# High-level Feature Proposal Breakdown

The following table describes the breakdown of features and their current ‘bucket’ in terms of “futures”, destined for v-next or already considered part of v-next.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature Proposal** | **Futures or ACT** | **v-next** | **Likely for v-next** |
| [Ajax and MVC](#_Ajax_and_MVC) |  | X |  |
| [DOM manipulation APIs](#_DOM_APIs) |  | X |  |
| [Accessibility](#_Accessibility) |  |  | X |
| [Client data and templating](#_Client_data_and) |  | X |  |
| [Client-side controls and behavior](#_Client-side_controls_and) | X |  |  |
| [OpenAjax](#_OpenAjax) |  | X |  |
| [Animation](#_Animation) |  |  | X |
| [Drag & Drop](#_Drag_&_Drop) |  |  | X |
| [JavaScript Tools](#_Provide_a_great) |  | X |  |

# Conclusion

This is just a brief summary of some exploratory work that the ASP.NET and Visual Web Developer teams are investigating. We are highly interested in your comments, and suggestions which will help us decide not only the feature sets that we invest in, but the designs that we embark upon.

So when will you start to see these improvements? We are likely to release sample code for the runtime improvements in a “Futures” form via the [ASP.NET CodePlex project](http://www.codeplex.com/aspnet). You can already see a model for this design process from the ASP.NET team in the manner of the ASP.NET MVC code now released via Codeplex ([www.codeplex.com/aspnet](http://www.codeplex.com/aspnet)) and the supporting CTPs and documentation. Details will be coming soon!

Many Thanks!

The Microsoft ASP.NET and the Visual Web Developer Teams.