This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

**Creating Custom AJAX Client Controls**

This overview shows you how to create a custom ASP.NET AJAX client control and use it in a page. In this overview you will learn how to do the following:

* Use the prototype design pattern in ECMAScript (JavaScript) to define a control class.
* Register a control as a class that derives from the [Sys.UI.Control](http://msdn.microsoft.com/en-us/library/bb310848.aspx) base class.
* Initialize the **Control** base class and invoke its methods.
* Create custom events that a page developer can bind to and handle.
* Use the client control in a page and bind to the control's events.

The overview provides an example of a complete client control that creates a button with hover behavior.

This overview focuses on client controls. There are three types of ASP.NET AJAX client component objects:

* Non-visual components that derive from the [Sys.Component](http://msdn.microsoft.com/en-us/library/bb397516.aspx) base class and that have no UI representation.
* Behaviors that derive from [Sys.UI.Behavior](http://msdn.microsoft.com/en-us/library/bb311020.aspx).
* Controls that derive from **Control**.

The following table summarizes the differences between components, behaviors, and controls.

|  |  |
| --- | --- |
| **Client component object types** | **Summary** |
| Components | * Derive from the **Component** base class. * Typically have no UI representation, such as a timer component that raises events at intervals but is not visible on the page. * Have no associated DOM elements. * Encapsulate client code intended to be reusable across applications. |
| Behaviors | * Derive from the **Behavior** base class, which extends the **Component** base class. * Extend the behavior of DOM elements, such as a watermarking behavior that can be attached to an existing text box. * Can create UI elements, although they do not typically modify the markup of the associated DOM element that they are attached to. * If assigned an ID, can be accessed directly from the DOM element through a custom attribute (expando). * Do not require an association with another client object, such as a class derived from the **Control** or **Behavior** classes. * Can reference either a control or a non-control HTML element in their [element](http://msdn.microsoft.com/en-us/library/bb397703.aspx) property. |
| Controls | * Derive from the **Control** base class, which extends the **Component** base class. * Represent a DOM element as a client object, typically changing the original DOM element's ordinary behavior to provide new functionality. For example, a menu control might read **li** items from a **ul** element as its source data, but not display a bulleted list. * Are accessed from the DOM element directly through the control expando. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifPrerequisites

To run the client control example provided in this topic, you will need:

* An AJAX-enabled ASP.NET Web site. If you have such a site already configured, you can use that site for this example. For more information about how to create a virtual directory or site, see [How to: Create and Configure Virtual Directories in IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/zwk103ab.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating Basic Functionality for a Custom ASP.NET AJAX Client Control

An ASP.NET AJAX client control represents a DOM element as a client object and extends a markup representation or provides additional functionality for the element. For example, a client control might extend an HTML element to react to mouse events by applying different CSS styles.

A client control encapsulates JavaScript code that is intended to be reusable across applications. By deriving from the **Control** base class, your custom control automatically inherits many built-in cross-browser features, which includes the following:

* The ability to add and remove event handlers for DOM elements associated with the control and for event handlers for the control itself.
* Automatic registration of the control as a disposable object that implements the [Sys.IDisposable](http://msdn.microsoft.com/en-us/library/bb310932.aspx) interface.
* The ability to raise notification events when properties are changed.
* The ability to perform batch processing of control property settings. This is more efficient in script size and processing time than handling all logic in individual property get and set accessors.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifImplementing a Client Control

The following table summarizes the steps for implementing a custom client control derived from **Control**. More detailed information about each step follows the table.

|  |  |
| --- | --- |
| **Step** | **Summary** |
| Define a client control class by using the prototype design pattern. | 1. Register the namespace for your control class. 2. Define the control's constructor function to receive a DOM element argument and define a prototype. 3. Register the control function as a class that is derived from the **Control** base class. |
| Initialize the control's base **Control** instance and pass the associated DOM element as an argument. | * In the control's constructor, call its inherited [Type.initializeBase](http://msdn.microsoft.com/en-us/library/bb397474.aspx) method and pass the DOM element received in the constructor argument to the base. |
| Expose any property accessors and optionally raise a [Sys.Component.propertyChanged](http://msdn.microsoft.com/en-us/library/bb311023.aspx) notification event. | * Expose properties in the component's prototype with get and set accessor methods. * For any properties where changed notification might be important, raise a **propertyChanged** notification event from the property's set accessor. |
| Override the [Sys.UI.Control.initialize](http://msdn.microsoft.com/en-us/library/bb397711.aspx) method to initialize any properties and event listeners. | If there are any properties or event listeners for the component or DOM elements to initialize, override the **initialize** method in the component's prototype. In the overridden method, do the following:   * Add any delegates to DOM events of the **window** object or of an element by calling the [Sys.UI.DomEvent.addHandler](http://msdn.microsoft.com/en-us/library/bb310798.aspx) method. * Set any initial DOM element properties. * Set any accessibility properties, such as the tab index on the DOM element or the control's **role** attribute. * Call the **initialize** method of the base class. |
| Override the [Sys.UI.Control.dispose](http://msdn.microsoft.com/en-us/library/bb311016.aspx) method to release resources, such as removing DOM event handlers. | If there are any resources to release before the control is disposed, override the **dispose** method in the component's prototype. In the overridden method, do the following:   * Stop any processes that the control might queue internally and disable any functionality that is exposed to callers. * Remove DOM event handlers before the control is disposed. By clearing any DOM element handlers that are set by your control, you remove any circular references to those DOM elements and release memory reserved for those elements. * Call the base **dispose** method. All code in the **dispose** method must be callable multiple times. For example, before you try to release a resource, verify that the resource has not already been released. |

**Defining a Control Class Using the Prototype Design Pattern**

An ASP.NET AJAX client class, which includes a control class, is defined in JavaScript using the prototype design pattern. For details, see [Creating a Client Component Class Using the Prototype Model](http://msdn.microsoft.com/en-us/library/bb386415.aspx).

A client control class must be derived from the **Control** base class. You register an ASP.NET AJAX client class as a class with the client application by using the **Type.registerClass** method. For more information, see [Type.registerClass Method](http://msdn.microsoft.com/en-us/library/bb383792.aspx).

**Initializing the Base Class**

The base **Control** object is initialized in the control's constructor. In the control's constructor, you invoke the inherited **initializeBase** method and pass the DOM element that was received in the constructor argument to the base class. Typically the **initializeBase** method is invoked before any other code runs in the constructor. When the **Control** base class is initialized, its methods are available to the control, and it automatically registers the control as a disposable object with the [Sys.Application](http://msdn.microsoft.com/en-us/library/bb310856.aspx) instance. For more information, see [Sys.IDisposable Interface](http://msdn.microsoft.com/en-us/library/bb310932.aspx).

The following example shows a constructor function for a control that derives from **Control**. The component's constructor calls the inherited **initializeBase** method.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl09_code');" \o "Copy Code)

Samples.SimpleControl = function(element)

{

Samples.SimpleControl.initializeBase(this, [element]);

}

**Defining Properties and Raising Property-changed Notifications**

You define properties in the client control's class that page developers can get and set. You can also raise **propertyChanged** notification events for your component's properties. Page developers who use your component can then bind to these events. An ASP.NET AJAX component derived from the **Component**, **Behavior**, or **Control** base class inherits the [Sys.Component.raisePropertyChanged](http://msdn.microsoft.com/en-us/library/bb397463.aspx) method, which you call in order to raise a **propertyChanged** event. For more information, see [Defining Custom Component Properties and Raising PropertyChanged Events](http://msdn.microsoft.com/en-us/library/bb398901.aspx).

**Initializing Properties and Event Listeners**

If your custom control must initialize any properties or event listeners, override the **initialize** method in the component's prototype. A client control that is derived from the **Control** base class typically binds any handlers to its DOM element events and sets the DOM element properties to initial values. As a final step, you call the base **initialize** method to enable the component's base class to complete initialization.

**Releasing Resources**

If your custom control must release resources before the control is disposed, override the **dispose** method and release the resources in the overridden method. This makes sure that the resources are released immediately before the control is disposed. Resources to release include handlers that are used to bind to DOM events. By verifying that any possible circular references between DOM elements and the component object are removed, you make sure that the object can be removed from memory. For more information, see [Releasing Component Resources](http://msdn.microsoft.com/en-us/library/bb398784.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifUsing a Control in a Page

To use a custom client control in an ASP.NET Web page, you do the following:

* Register the client control's script library in the Web page.
* Create a client control instance.

The following sections provide details about these steps.

**Registering A Control's Script Library in the Web Page**

You can register scripts that are required for a client control on the page with a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control, either declaratively or programmatically.

The following example shows the declarative markup for a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control that registers a control script.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl02_code');" \o "Copy Code)

<form id="form1" runat="server">

<asp:ScriptManager runat="server" ID="ScriptManager01">

<scripts>

<asp:ScriptReference path="HoverButton.js" />

</scripts>

</asp:ScriptManager>

</form>

The **asp:ScriptManager** element contains an **asp:ScriptReference** element inside a **scripts** node. The **path** attribute of the **asp:ScriptReference** element references the path of the HoverButton.js file that defines a control class. For more information, see [Dynamically Assigning Script References](http://msdn.microsoft.com/en-us/library/bb398991.aspx) and the [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) class overview.

|  |
| --- |
| **Description: NoteNote** |
| All standalone script files that will be registered with the [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control must call the **notifyScriptLoaded** method to notify the application that the script has finished loading. Scripts that are embedded in an assembly should not call this method most of the time. For more information, see [Sys.Application.notifyScriptLoaded Method](http://msdn.microsoft.com/en-us/library/bb310952.aspx). |

As an alternative to registering script files by using the [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control, you can manage client components by using a custom server control that implements the [IScriptControl](http://msdn.microsoft.com/en-us/library/system.web.ui.iscriptcontrol.aspx) interface. A custom server control can automatically register the required component scripts and expose declarative markup for setting for component properties and event bindings. This makes it easier for a page developer to use your custom control. For more information, see the [IScriptControl](http://msdn.microsoft.com/en-us/library/system.web.ui.iscriptcontrol.aspx) class overview.

**Creating a Custom Control Instance**

You instantiate a custom client control by calling the [Sys.Component.create](http://msdn.microsoft.com/en-us/library/bb310863.aspx) method or the [$create](http://msdn.microsoft.com/en-us/library/bb397487.aspx) shortcut during the [Sys.Application.init](http://msdn.microsoft.com/en-us/library/bb397532.aspx) event. The following table describes the parameters that you pass to the **$create** method when you create a client control.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *type* | The component type. |
| *properties* | A JSON object that contains a component ID value and optionally any initial property name/value pairs. |
| *events* | An optional JSON object that contains event name and event/handler binding pairs. |
| *references* | An optional JSON object that contains references to associated components, passed as component name/ID pairs. |
| *element* | The DOM element to associate with the control. |

This following example shows how to instantiate a control instance by calling the **$create** method.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl14_code');" \o "Copy Code)

$create(Demo.HoverButton, {text: 'A HoverButton Control',element: {style: {fontWeight: "bold", borderWidth: "2px"}}}, {click: start, hover: doSomethingOnHover, unhover: doSomethingOnUnHover},null, $get('Button1'));

For more information, see [Sys.Component.create Method](http://msdn.microsoft.com/en-us/library/bb310863.aspx) and [Sys.Component $create Method](http://msdn.microsoft.com/en-us/library/bb397487.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating the Custom HoverButton Control

In this section you will create a simple custom client control named HoverButton that extends the **Control** base class, and then use the control in a page. The HoverButton control intercepts the **click**, **focus**, and **mouseover** events of an associated HTML **button** element. It also provides the controls with events that are bindable through the **$create** method. A page developer who uses the HoverButton control can bind to the control's hover event.

**To create the code for the HoverButton control**

1. In the root directory of an AJAX-enabled ASP.NET Web site, create a file named HoverButton.js.
2. Add the following code to the file:

JScript

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl77_ctl00_ctl00_code');" \o "Copy Code)

Type.registerNamespace("Demo");

// Constructor

Demo.HoverButton = function(element) {

Demo.HoverButton.initializeBase(this, [element]);

this.\_clickDelegate = null;

this.\_hoverDelegate = null;

this.\_unhoverDelegate = null;

}

Demo.HoverButton.prototype = {

// text property accessors.

get\_text: function() {

return this.get\_element().innerHTML;

},

set\_text: function(value) {

this.get\_element().innerHTML = value;

},

// Bind and unbind to click event.

add\_click: function(handler) {

this.get\_events().addHandler('click', handler);

},

remove\_click: function(handler) {

this.get\_events().removeHandler('click', handler);

},

// Bind and unbind to hover event.

add\_hover: function(handler) {

this.get\_events().addHandler('hover', handler);

},

remove\_hover: function(handler) {

this.get\_events().removeHandler('hover', handler);

},

// Bind and unbind to unhover event.

add\_unhover: function(handler) {

this.get\_events().addHandler('unhover', handler);

},

remove\_unhover: function(handler) {

this.get\_events().removeHandler('unhover', handler);

},

// Release resources before control is disposed.

dispose: function() {

var element = this.get\_element();

if (this.\_clickDelegate) {

Sys.UI.DomEvent.removeHandler(element, 'click', this.\_clickDelegate);

delete this.\_clickDelegate;

}

if (this.\_hoverDelegate) {

Sys.UI.DomEvent.removeHandler(element, 'focus', this.\_hoverDelegate);

Sys.UI.DomEvent.removeHandler(element, 'mouseover', this.\_hoverDelegate);

delete this.\_hoverDelegate;

}

if (this.\_unhoverDelegate) {

Sys.UI.DomEvent.removeHandler(element, 'blur', this.\_unhoverDelegate);

Sys.UI.DomEvent.removeHandler(element, 'mouseout', this.\_unhoverDelegate);

delete this.\_unhoverDelegate;

}

Demo.HoverButton.callBaseMethod(this, 'dispose');

},

initialize: function() {

var element = this.get\_element();

if (!element.tabIndex) element.tabIndex = 0;

if (this.\_clickDelegate === null) {

this.\_clickDelegate = Function.createDelegate(this, this.\_clickHandler);

}

Sys.UI.DomEvent.addHandler(element, 'click', this.\_clickDelegate);

if (this.\_hoverDelegate === null) {

this.\_hoverDelegate = Function.createDelegate(this, this.\_hoverHandler);

}

Sys.UI.DomEvent.addHandler(element, 'mouseover', this.\_hoverDelegate);

Sys.UI.DomEvent.addHandler(element, 'focus', this.\_hoverDelegate);

if (this.\_unhoverDelegate === null) {

this.\_unhoverDelegate = Function.createDelegate(this, this.\_unhoverHandler);

}

Sys.UI.DomEvent.addHandler(element, 'mouseout', this.\_unhoverDelegate);

Sys.UI.DomEvent.addHandler(element, 'blur', this.\_unhoverDelegate);

Demo.HoverButton.callBaseMethod(this, 'initialize');

},

\_clickHandler: function(event) {

var h = this.get\_events().getHandler('click');

if (h) h(this, Sys.EventArgs.Empty);

},

\_hoverHandler: function(event) {

var h = this.get\_events().getHandler('hover');

if (h) h(this, Sys.EventArgs.Empty);

},

\_unhoverHandler: function(event) {

var h = this.get\_events().getHandler('unhover');

if (h) h(this, Sys.EventArgs.Empty);

}

}

Demo.HoverButton.registerClass('Demo.HoverButton', Sys.UI.Control);

// Since this script is not loaded by System.Web.Handlers.ScriptResourceHandler

// invoke Sys.Application.notifyScriptLoaded to notify ScriptManager

// that this is the end of the script.

if (typeof(Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();

**Code Discussion**

The code registers the Demo namespace by calling the [Type.registerNamespace](http://msdn.microsoft.com/en-us/library/bb397723.aspx) method. The constructor invokes the inherited **initializeBase** method so that the **Control** base class methods are available. The initialized base class in turn registers the Demo.HoverButton instance with the client application as a disposable object.

In the prototype, the code declares the public events click, hover, and unhover. The page developer can add and remove handlers that listen for those events. These methods in turn add or remove the specified handler through the control's event-handler collection. You add and remove handlers in your control class through the control's [Sys.EventHandlerList](http://msdn.microsoft.com/en-us/library/bb383996.aspx) object. The **EventHandlerList** object contains a collection of the control's event handlers through the inherited [Sys.Component.events](http://msdn.microsoft.com/en-us/library/bb383770.aspx) property. In the example, the code invokes the [Sys.EventHandlerList.addHandler](http://msdn.microsoft.com/en-us/library/bb384007.aspx) and [Sys.EventHandlerList.removeHandler](http://msdn.microsoft.com/en-us/library/bb310827.aspx) methods of the returned **EventHandlerList** object in order to add or remove the handlers.

The HoverButton class overrides the base [dispose](http://msdn.microsoft.com/en-us/library/bb311016.aspx) method to safely dispose any control resources (such as handlers for DOM events) before the control is disposed. Finally, the code calls the base **dispose** method to enable the application to release the control.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifUsing the HoverButton Control in a Web Page

In this section, you will learn how to create a control instance by using client script in a Web page.

**To create a page to use the HoverButton control**

1. In the application root directory where you put the HoverButton.js file, create a file named DemoHoverButton.aspx.
2. Add the following markup and code to the file:

JScript

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl78_ctl00_ctl00_code');" \o "Copy Code)

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

<html >

<head id="Head1" runat="server">

<style type="text/css">

button {border: solid 1px black}

#HoverLabel {color: blue}

</style>

<title>Control Demo</title>

</head>

<body>

<form id="form1" runat="server">

<div id="ResultDisplay"></div>

<asp:ScriptManager runat="server" ID="ScriptManager01">

<scripts>

<asp:ScriptReference Path="HoverButton.js" />

</scripts>

</asp:ScriptManager>

<script type="text/javascript">

var app = Sys.Application;

app.add\_init(applicationInitHandler);

function applicationInitHandler(sender, args) {

$create(Demo.HoverButton, {text: 'A HoverButton Control',element: {style: {fontWeight: "bold", borderWidth: "2px"}}}, {click: start, hover: doSomethingOnHover, unhover: doSomethingOnUnHover},null, $get('Button1'));

}

function doSomethingOnHover(sender, args) {

hoverMessage = "The mouse is over the button."

$get('HoverLabel').innerHTML = hoverMessage;

}

function doSomethingOnUnHover(sender, args) {

$get('HoverLabel').innerHTML = "";

}

function start(sender, args) {

alert("The start function handled the HoverButton click event.");

}

</script>

<button type="button" id="Button1"></button>&nbsp;

<div id="HoverLabel"></div>

</form>

</body>

</html>

**Code Discussion**

The DemoHoverButton.aspx file is an ASP.NET Web page that hosts the custom control. In the page, the functions that are bound to the custom control are defined in the **script** element. In the [Sys.Application.init](http://msdn.microsoft.com/en-us/library/bb397532.aspx) event handler, the HoverButton control is instantiated in client script by calling the **$create** method. The code passes the following arguments to the **$create** method:

* The *type* argument contains the Demo.HoverButton class that you created earlier.
* The *properties* argument contains a JSON object that contains the required control ID value, followed by property name-value pairs that specify property names with initial values.
* The *events* argument contains an object that contains event names paired with their handlers.

In the [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control, the **path** attribute of the **asp:ScriptReference** node references the path of the HoverButton.js file that defines the Demo.HoverButton control class.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifSetting DOM-element Event Handlers and Component Event Handlers

AJAX functionality in ASP.NET includes classes that provide standardized event management for components and for DOM elements. You manage your control's events by using members of the [Sys.EventHandlerList](http://msdn.microsoft.com/en-us/library/bb383996.aspx) class, such as **addHandler** and **removeHandler**. For more information, see the [Sys.EventHandlerList](http://msdn.microsoft.com/en-us/library/bb383996.aspx) class overview.

You manage event handlers for DOM elements or for **window** object events by using static methods of the [Sys.UI.DomEvent](http://msdn.microsoft.com/en-us/library/bb383775.aspx) class, [addHandler](http://msdn.microsoft.com/en-us/library/bb310798.aspx) or [removeHandler](http://msdn.microsoft.com/en-us/library/bb310935.aspx). For more information, see the [Sys.UI.DomEvent](http://msdn.microsoft.com/en-us/library/bb383775.aspx) class overview.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAccessing DOM-element Properties

The **Sys.UI.DomElement** class contains members that enable you to add, remove, and toggle CSS class associations for client controls and elements. These members also provide standardized access to DOM element properties. For more information, see [Sys.UI.DomElement Class](http://msdn.microsoft.com/en-us/library/bb383788.aspx).