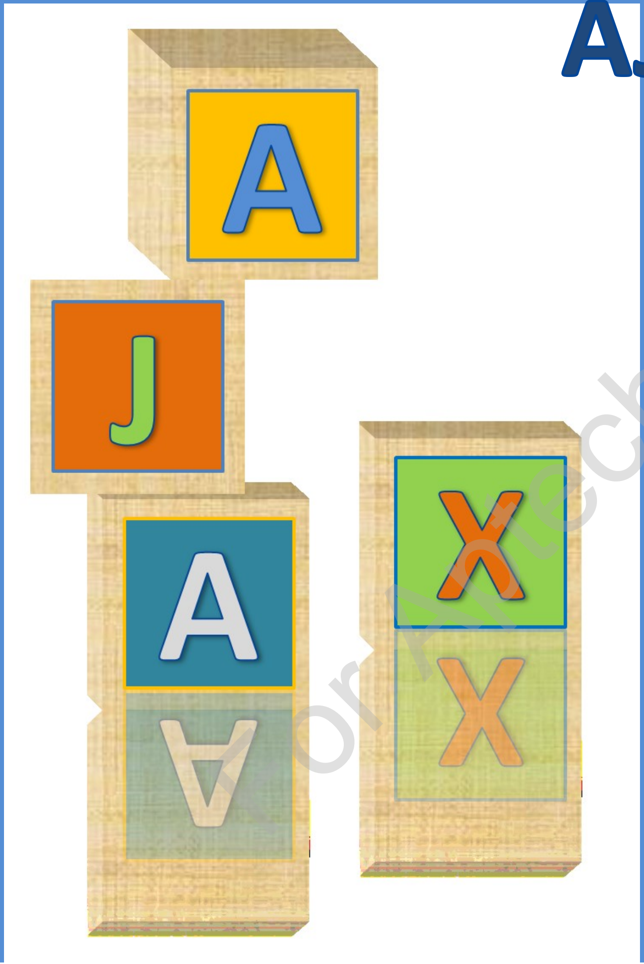
# Programming ASP.NET AJAX

## Session: 3

## AJAX Client Library



## Objectives

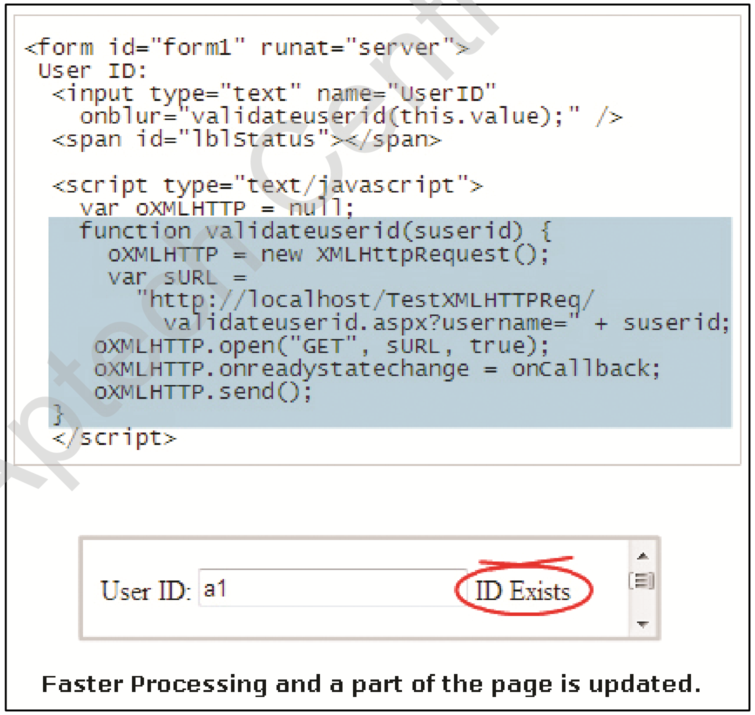
AJAX

- Describe Microsoft AJAX Client Library
- Explain the `Type` class
- Explain JavaScript Base Type Extensions
- Explain the advanced JavaScript Base Type Extensions
- Explain `Sys` Namespace and `Sys.UI` Namespace

## JavaScript and Web Applications

AJAX

- Microsoft AJAX Library provides a set of tools used for client development.
- JavaScript provides the following benefits:
  - Allows access to server resources
  - Results are processed quickly
  - Enables smooth Web page interactivity
- Use of JavaScript and AJAX results in rich Web-based applications as shown in the figure.



```
<form id="form1" runat="server">
 User ID:
  <input type="text" name="UserID"
    onblur="validateuserid(this.value);" />
  <span id="lblStatus"></span>

  <script type="text/javascript">
    var oXMLHTTP = null;
    function validateuserid(suserid) {
      oXMLHTTP = new XMLHttpRequest();
      var sURL =
        "http://localhost/TestXMLHTTPReq/
          validateuserid.aspx?username=" + suserid;
     oXMLHTTP.open("GET", sURL, true);
     oXMLHTTP.onreadystatechange = onCallback;
     oXMLHTTP.send();
  }
  </script>
```

Faster Processing and a part of the page is updated.

## Microsoft AJAX Client Library

- Microsoft AJAX Client Library provides a set of tools used for client development.
- Asynchronous requests are made using `XMLHttpRequest` object.
- Supports the following features:

**Application Model**

- Includes the `Application` object, responsible for hosting and disposing the client components.

**Components**

- Reusable components can be created by using JavaScript.

**JavaScript Extensions**

- An object model to support reflection and object-oriented components is provided.

**Compatibility**

- Compatibility ensures that the codes run on all supported browsers and the user can access the DOM independently.

**Application Services**

- Authentication, membership, and profile providers on the client-side are provided.

**Partial Rendering**

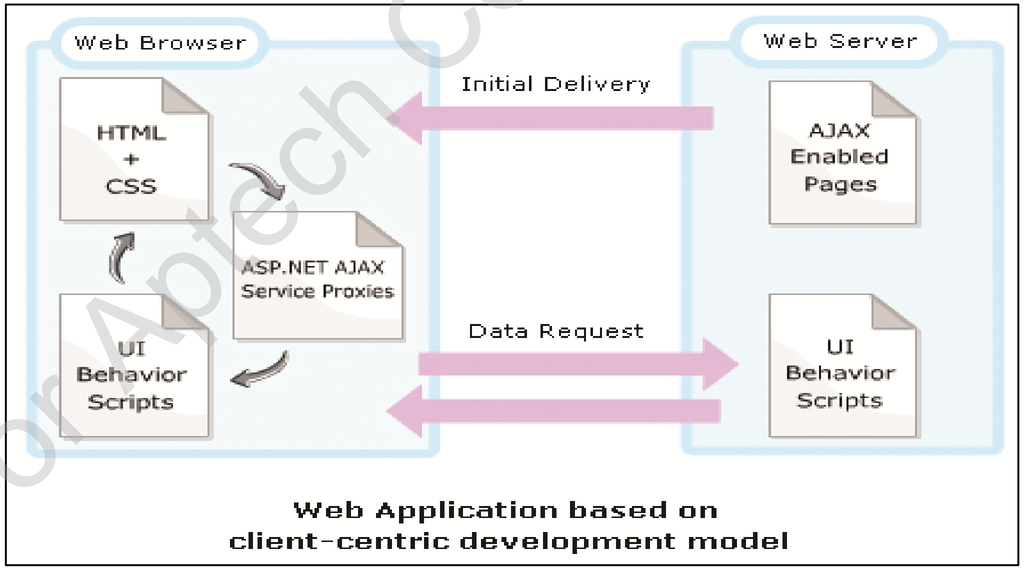
- Partial page rendering is made possible with the use of the `UpdatePanel` control.

## Client-centric Development Model

AJAX

- Implemented on the client-side.
- Helps display interactive Web pages.
- Helps create UI with the help of DHTML and JavaScript.

1. A rich interactive application is delivered when a page is loaded.
2. Only the relevant data is retrieved to update the page.
3. This results in increased interaction between the user and browser application as shown in the figure.



Web Application based on client-centric development model

## Type System

AJAX

- Provides object-oriented features to JavaScript objects and AJAX enabled ASP.NET applications.
- Helps create functionalities in a modular fashion and improve maintainability of client applications.
- Some of the capabilities added to JavaScript are as follows:

Classes

Namespaces

Inheritance

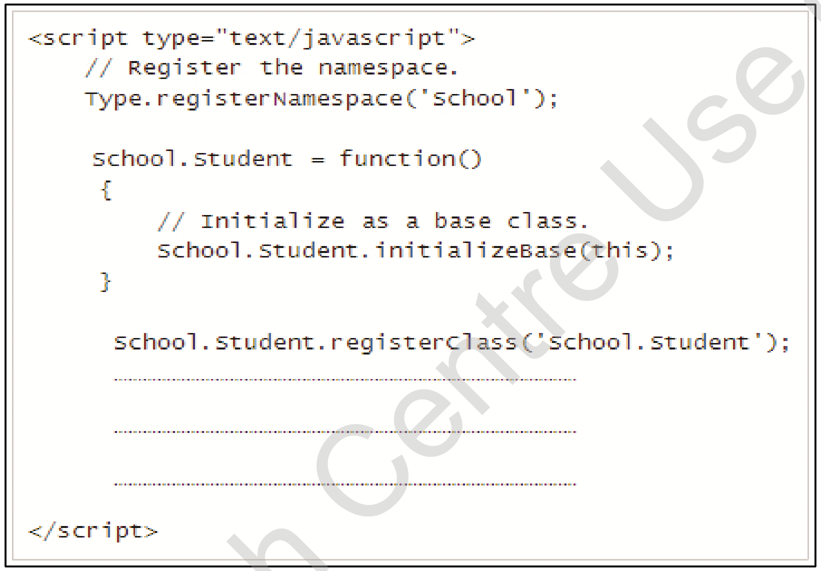
Interfaces

Enumerations

Reflections

## Type Class

- Allows addition of namespaces, classes, interfaces, and inheritance to JavaScript code as shown in the figure.



```
<script type="text/javascript">
    // Register the namespace.
    Type.registerNamespace('School');

     School.Student = function()
     {
         // Initialize as a base class.
         School.Student.initializeBase(this);
     }

      School.Student.registerClass('School.Student');
      ..................................................

      ..................................................

      ..................................................

</script>
```

- Following syntax and code snippet registers a namespace.

**Syntax**

```
Type.registerNamespace(string);
```

**Code Snippet**

```
Type.registerNamespace('School');
```

## Type Class Members

- The `Type` class provides a `registerClass()` method to create a class in JavaScript as shown in the following table.

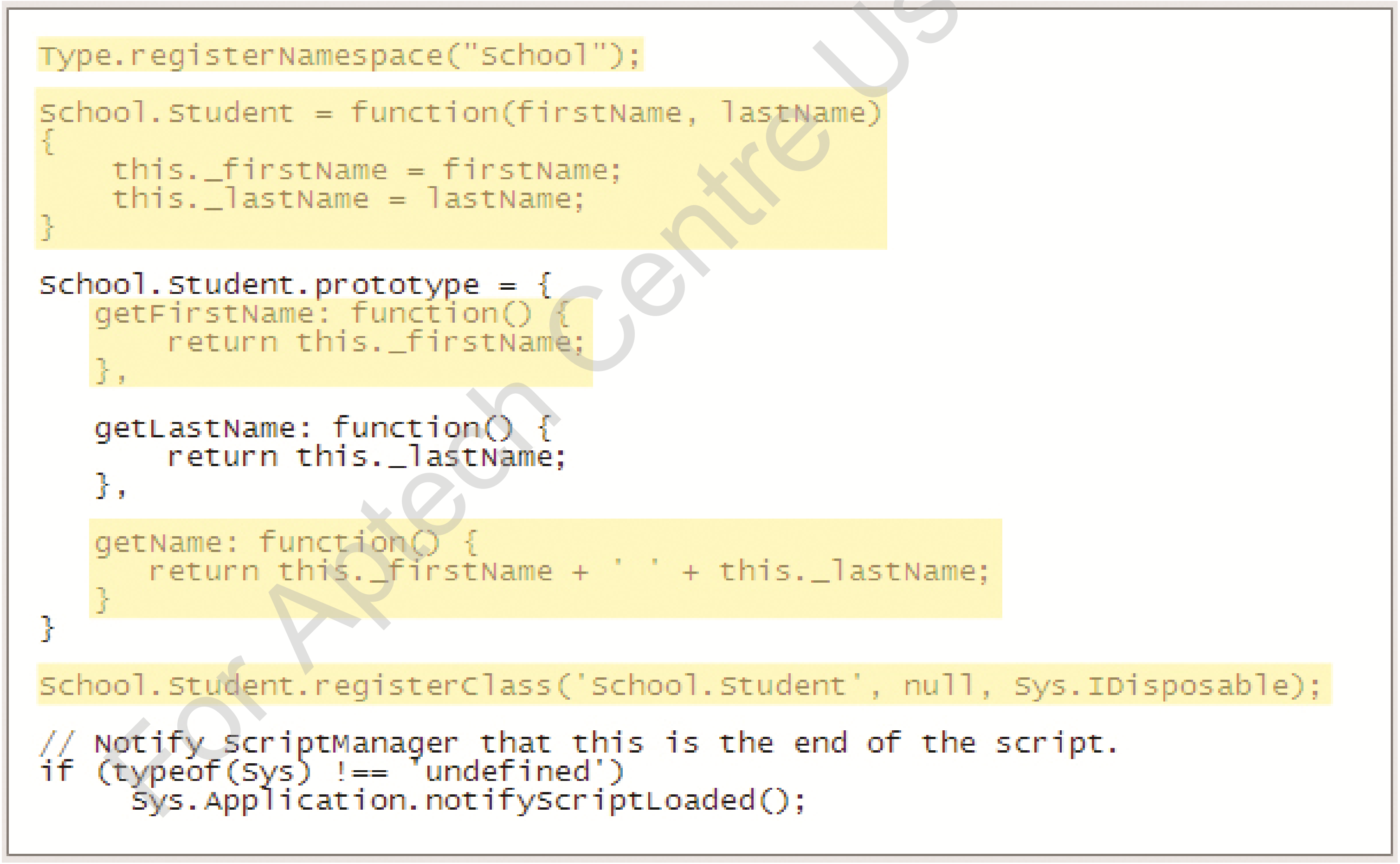
| Method | Description |
|---|---|
| `createCallback` | Creates a callback method to set up a handler for a DOM event that retains a parameter. |
| `getBaseType` | Retrieves the base type of an instance. |
| `getName` | Retrieves the name of the instance type. |
| `isClass` | Retrieves a value to indicate whether the specified type is a class. |
| `isInstanceOfType` | Specifies whether the object is an instance of a specified type or of one of its derived types. |
| `Parse` | Retrieves an instance of the type that is specified by a type name. |
| `registerClass` | Registers a class. |
| `registerNamespace` | Registers and creates a namespace. |

## Type Class Example 1-5

AJAX

### Step 1: School.js

- An example to create a separate JavaScript file (.js) to define classes and its members as shown in the figure.



```
Type.registerNamespace("School");

School.Student = function(firstName, lastName)
{
    this._firstName = firstName;
    this._lastName = lastName;
}

School.Student.prototype = {
   getFirstName: function() {
       return this._firstName;
   },

   getLastName: function() {
       return this._lastName;
   },

   getName: function() {
      return this._firstName + ' ' + this._lastName;
   }
}

School.Student.registerClass('School.Student', null, Sys.IDisposable);

// Notify ScriptManager that this is the end of the script.
if (typeof(Sys) !== 'undefined')
     Sys.Application.notifyScriptLoaded();
```

## Type Class Example 2-5

AJAX

### Step 1: School.js

```
Type.registerNamespace("School");
```

```
School.Student = function(firstName, lastName)
{
    this._firstName = firstName;
    this._lastName = lastName;
}
```

```
getFirstName: function() {
        return this._firstName;
   }
```

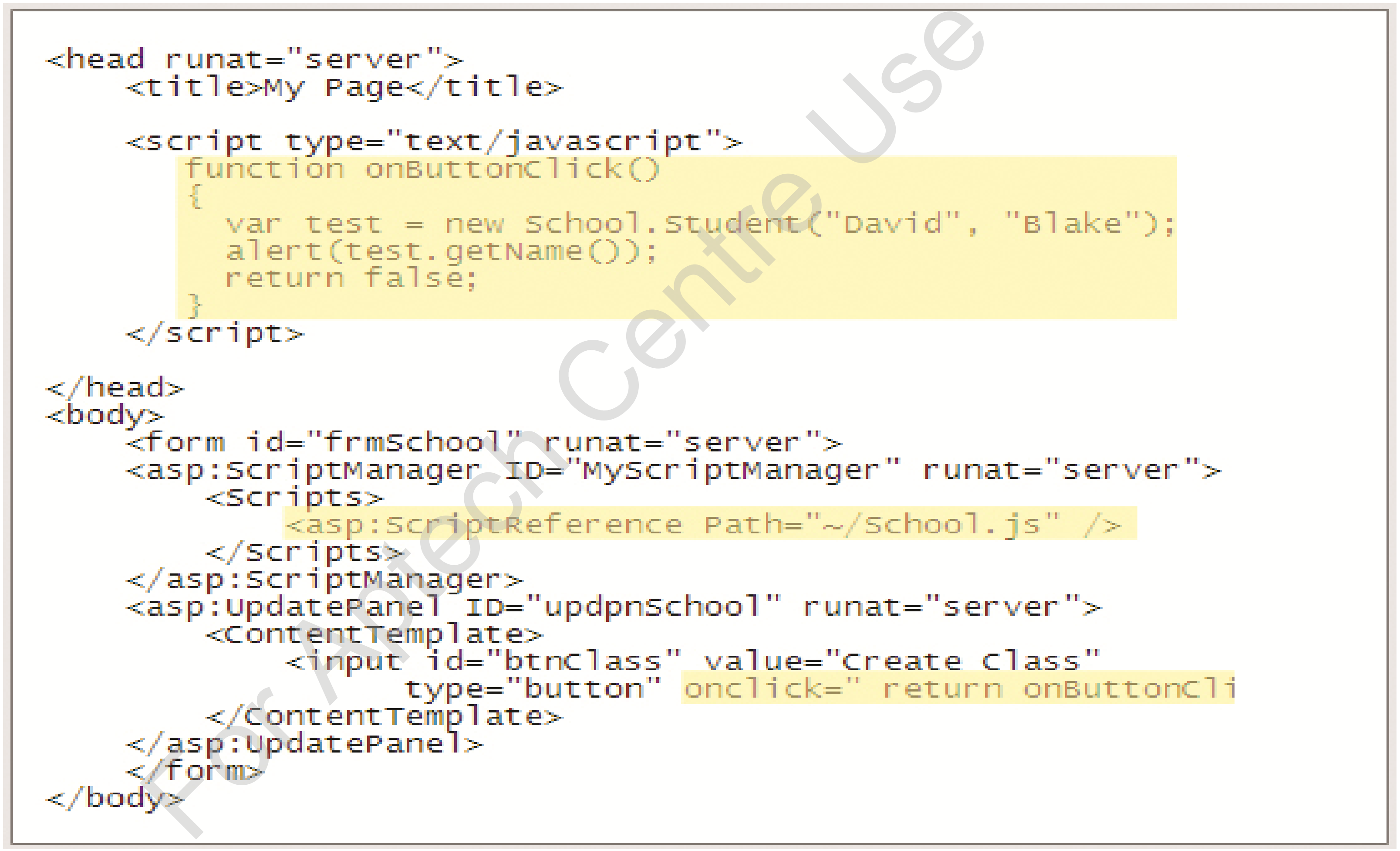
```
getName: function() {
      return this._firstName + ' ' + this._lastName;
}
```

```
School.Student.registerClass('School.Student', null,
Sys.IDisposable);
```

## Type Class Example 3-5

### Step 2: Default.aspx

- The `Default.aspx` file is shown in the figure.



```
<head runat="server">
    <title>My Page</title>

    <script type="text/javascript">
      function onButtonClick()
      {
        var test = new School.Student("David", "Blake");
        alert(test.getName());
        return false;
      }
    </script>

</head>
<body>
    <form id="frmSchool" runat="server">
    <asp:ScriptManager ID="MyScriptManager" runat="server">
        <Scripts>
            <asp:ScriptReference Path="~/School.js" />
        </Scripts>
    </asp:ScriptManager>
    <asp:UpdatePanel ID="updpnSchool" runat="server">
        <ContentTemplate>
            <input id="btnClass" value="Create Class"
                  type="button" onclick=" return onButtonCli
        </ContentTemplate>
    </asp:UpdatePanel>
    </form>
</body>
```

## Type Class Example 4-5

AJAX

### Step 2: Default.aspx

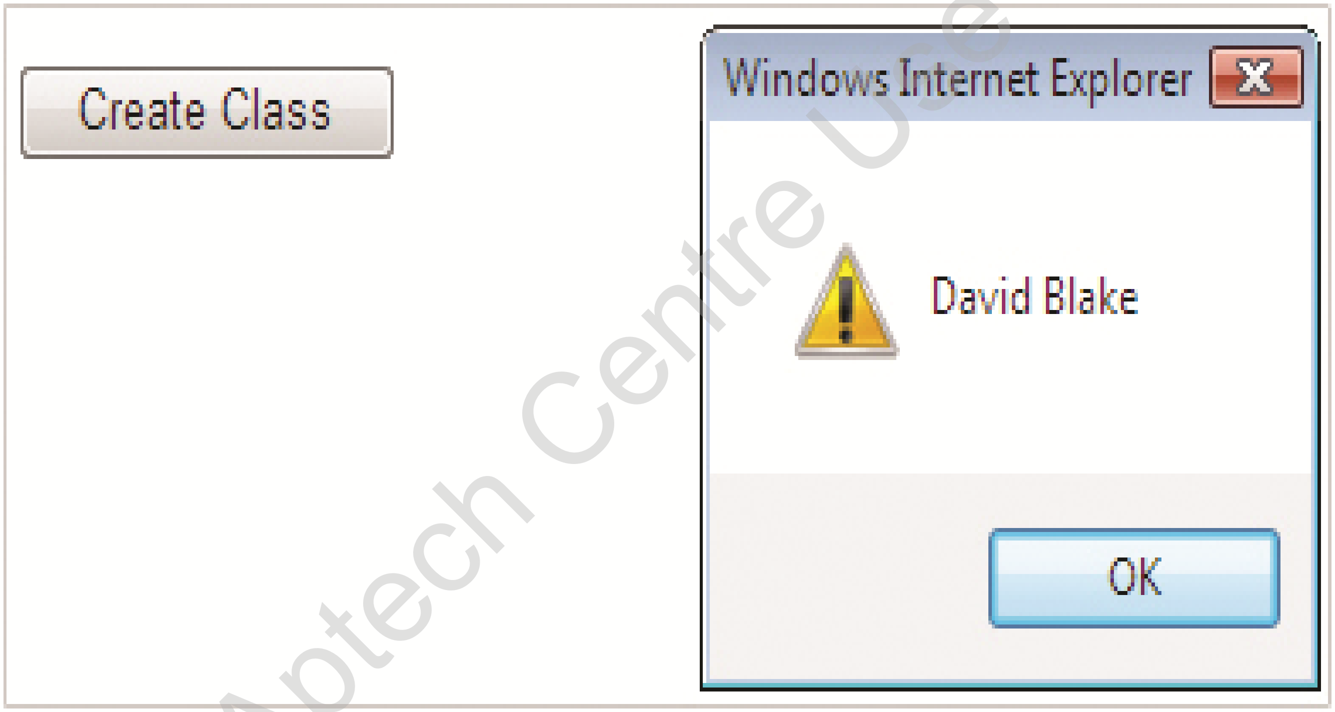
```
function onButtonClick()
{
        var test = new School.Student("David", "Blake");
        alert(test.getName());
        return false;
}
```

```
<asp:ScriptReference Path="~/School.js" />
```

```
onclick=" return onButtonClick()"
```

## Type Class Example 5-5

### Output



David Blake

**Output**

## Base Type Extensions 1-5

### Object Type Extensions

- The built-in JavaScript object named `Object` gets additional functionality from Object Type Extensions as shown in the syntax and the code snippet.
- Following table lists the functions about a typed instance.

**Syntax**

```
var myVar = new Object();
```

**Code Snippet**

```
<script type="text/javascript">
   ………………
   ………………
   var test = new
School.Student("David", "Blake");


   var type =
Object.getTypeName(test);
   alert(type);
   ………………
   ………………
</script>
```

| Function | Description |
|---|---|
| `getType` | Retrieves the type of an object instance. |
| `getTypeName` | Retrieves a string that specifies the run-time type name of an object. |

## Base Type Extensions 2-5

AJAX

### Array Type Extensions

- Extension of the `Array` base type allows user to interact with JavaScript arrays. Following table lists the functions.

| Function | Description |
|---|---|
| `add` | Adds an element to the end of an `Array` object. |
| `clear` | Removes all elements from an `Array` object. |
| `forEach` | Loops through the `Array` object and executes code for each of its elements. |
| `indexOf` | Returns the index of the specified element of an `Array` object. |
| `insert` | Inserts an element at the specified location in an `Array` object. |
| `parse` | Constructs an `Array` object from a string representation. |
| `remove` | Removes the first occurrence of the element in an `Array` object. |

## Base Type Extensions 3-5

AJAX

- Following code snippet creates an Array Type.

Code Snippet

```
function createArray()
{
   // Creates an array and stores 5 values in it.
   var myArray = new Array("David","Blake","John","Susan","Steffy");

   // Displaying the values of the array.
   for (i=0; i < myArray.length; i++)
   {
      alert(myArray[i]);
   }
   // Adding one more element at the end.
   Array.add(myArray,"Steve");
   // Finding the element, John and displaying its index
   alert(Array.indexOf(myArray,"John",0));
}
```

## Base Type Extensions 4-5

### Date Type Extensions

- Adds functionality to the JavaScript `Date` object as shown in the code snippet.

**Code Snippet**

```
var date = new Date();


alert(date.format("MM-dd-yyyy"));
alert(date.localeFormat("dd-MMM-yyyy"));
```

- Following table lists the functions.

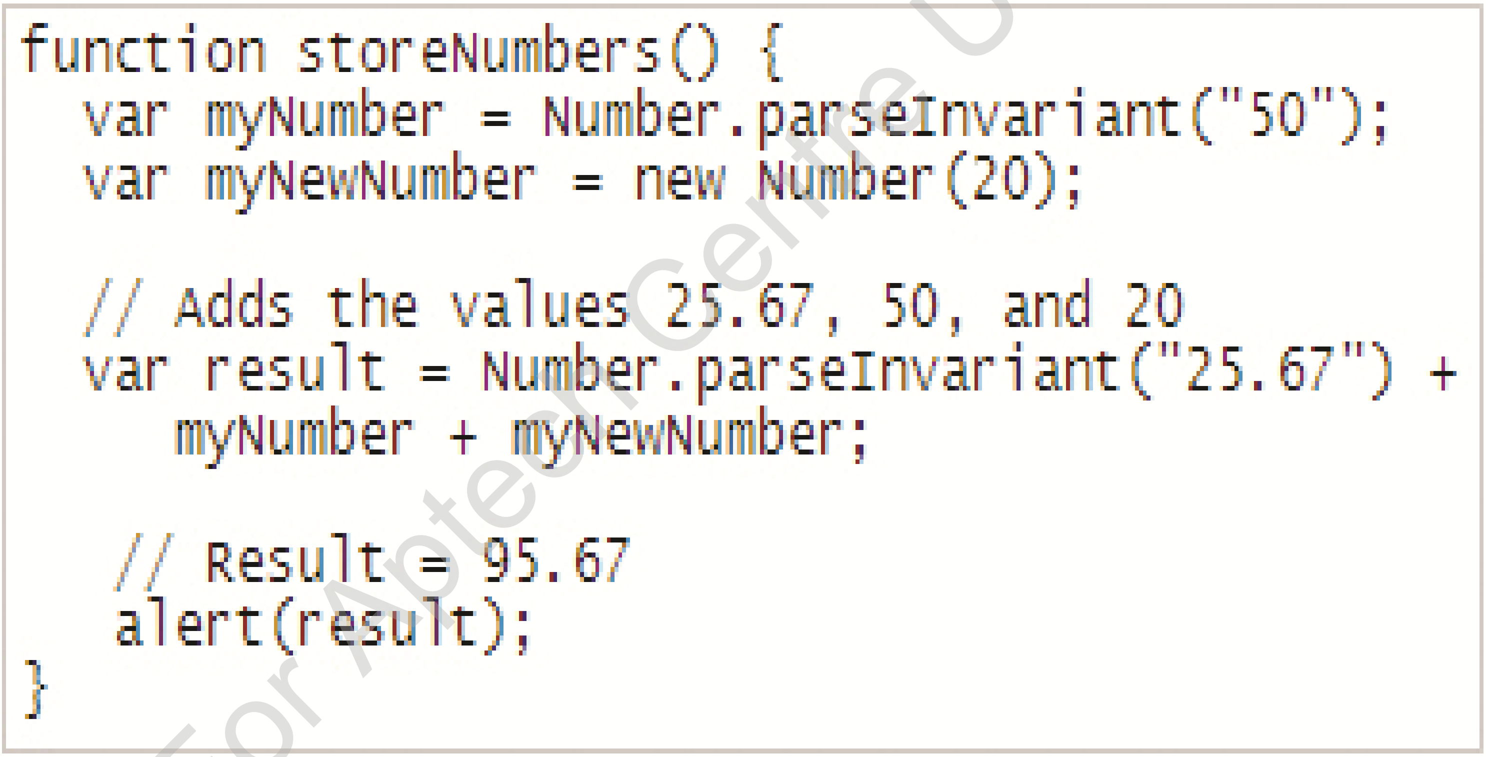
| Function | Description |
|---|---|
| `format` | Formats a date using the invariant culture. |
| `localeFormat` | Formats a date using the current culture. |
| `parseLocale` | Constructs a date from a locale-specific string by using the current culture. |
| `parseInvariant` | Constructs a date from a string by using the invariant culture. |

# Base Type Extensions 5-5

AJAX

## Number Type Extensions

- Adds functionality to the JavaScript `Number` object.
- Has the same functions as the `Date` object as shown in the figure.

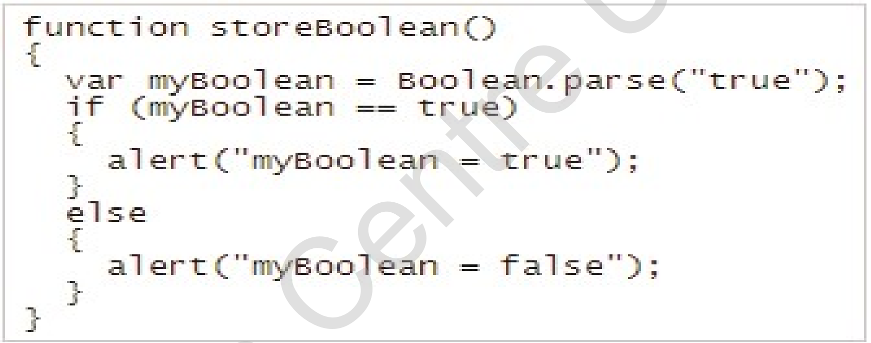


```
function storeNumbers() {
  var myNumber = Number.parseInvariant("50");
  var myNewNumber = new Number(20);

  // Adds the values 25.67, 50, and 20
  var result = Number.parseInvariant("25.67") +
     myNumber + myNewNumber;

   // Result = 95.67
   alert(result);
}
```

## Boolean Type Extensions

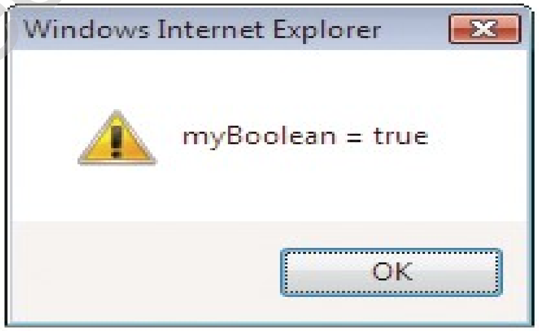
AJAX

- Adds functionality to the JavaScript `Boolean` object.
- The `parse()` function of this extension converts a string representation of a logical value to its `Boolean` equivalent.
- The `value` argument of this function must be a string representation of the `Boolean` value containing either `true` or `false` as shown in the figure.



```
function storeBoolean()
{
   var myBoolean = Boolean.parse("true");
   if (myBoolean == true)
   {
      alert("myBoolean = true");
   }
   else
   {
      alert("myBoolean = false");
   }
}
```

parse() function



```
myBoolean = true
```

Output

## Error Type Extensions

AJAX

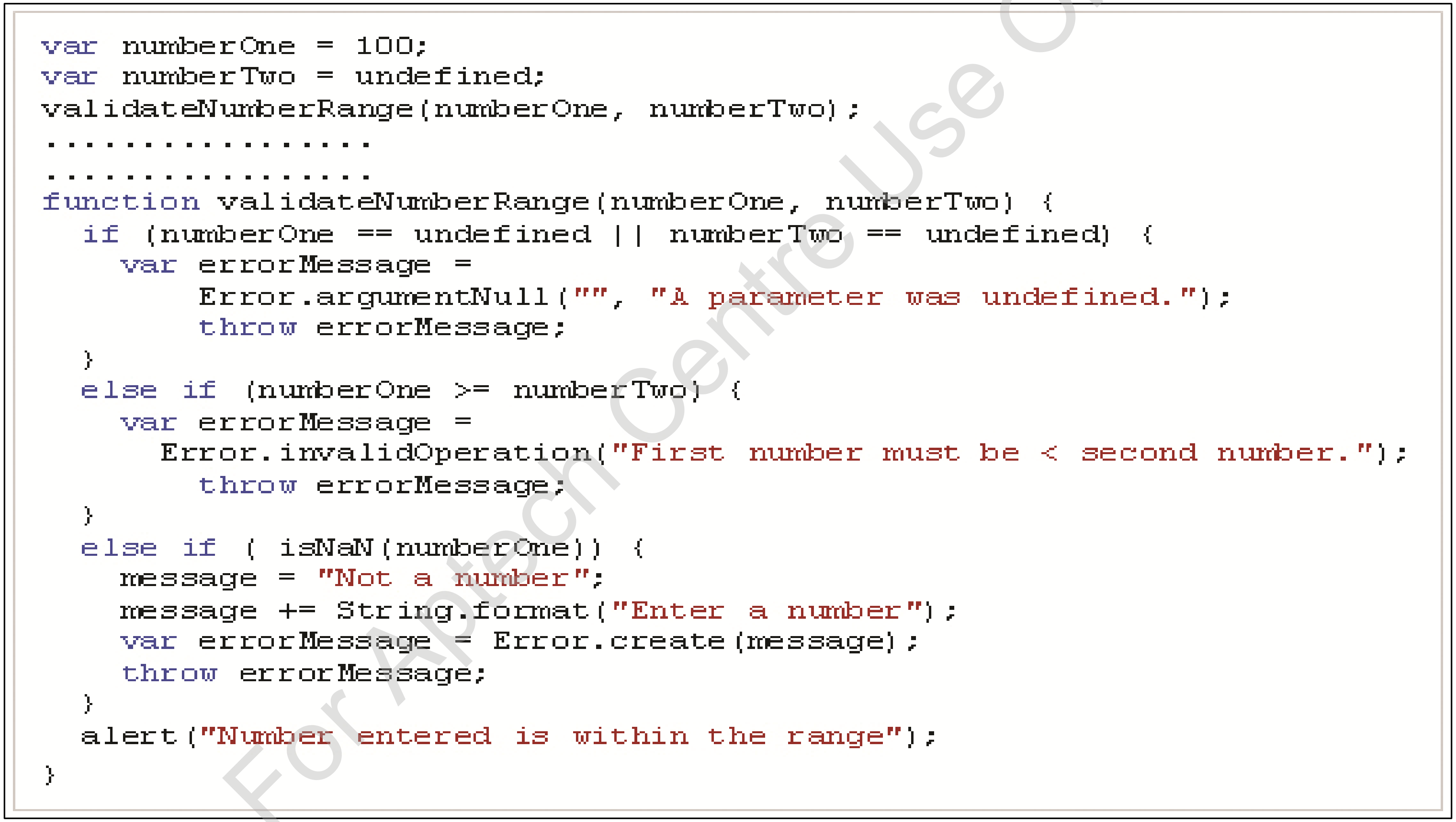
- Adds functionality to the built-in JavaScript `Error` object. Following table lists the names.

| Name | Description |
|---|---|
| `argument` | This function constructs an `Error` object which represents the `Sys.ArgumentException` exception. |
| `argumentNull` | This function constructs an `Error` object which represents the `Sys.ArgumentNullException` exception. |
| `argumentUndefined` | This function constructs an `Error` object which represents the `Sys.ArgumentUndefinedException` exception. |
| `create` | This function constructs an `Error` object that has additional error information. |
| `invalidOperation` | This function constructs an `Error` object which represents the `Sys.InvalidOperationException` exception. |
| `message` | This field represents the description of the error. |
| `name` | This field represents the name that identifies the error. |

## Error Type - Example

AJAX

- Following figure displays an example of error type.



```
var numberOne = 100;
var numberTwo = undefined;
validateNumberRange(numberOne, numberTwo);
..................
..................
function validateNumberRange(numberOne, numberTwo) {
  if (numberOne == undefined || numberTwo == undefined) {
    var errorMessage =
        Error.argumentNull("", "A parameter was undefined.");
        throw errorMessage;
  }
  else if (numberOne >= numberTwo) {
    var errorMessage =
      Error.invalidOperation("First number must be < second number.");
        throw errorMessage;
  }
  else if ( isNaN(numberOne)) {
    message = "Not a number";
    message += String.format("Enter a number");
    var errorMessage = Error.create(message);
    throw errorMessage;
  }
  alert("Number entered is within the range");
}
```

## String Type Extensions

AJAX

- Adds functionality to the built-in JavaScript `String` object. Following table lists the functions.

| Function | Description |
|---|---|
| `format` | Replaces each format item in a `String` object with the text representation of the corresponding object values. |
| `trim` | Removes the white space from beginning and end from the `String` object instance. |
| `trimEnd` | Removes the trailing white space from the `String` object instance. |
| `trimStart` | Removes leading white space from the `String` object instance. |

## Sys Namespace 1-3

AJAX

- Is the root namespace in the Microsoft AJAX Library and lists the classes in the following table.

| Class | Description |
|---|---|
| `Application` | Provides an object that manages client components of the application, and exposes client events. |
| `ApplicationLoadEventArgs` | Used by the `Application` class to hold event arguments for the `load` event. |
| `Component` | Provides the base class for ASP.NET AJAX client controls, behaviors, and non-visual components on the page. |
| `CultureInfo` | Represents a culture definition applied to objects that accepts a culture-related setting. |
| `Debug` | Provides tracing and debugging functionality to client JavaScript code. |
| `EventArgs` | Provides a base class for classes that are used by event sources to pass event argument information. |
| `StringBuilder` | Provides a mechanism to join a sequence of strings. |

## Sys Namespace 2-3

### Application Class

- Is responsible for raising events such as `pageInit`, `pageLoad`, and `pageUnload` for the page and for managing disposal of the registered components. Following table lists the names.

| Name | Description |
|---|---|
| `addComponent` | Registers and initializes a component with the application. |
| `dispose` | Release resources and dependencies held by the client application. |
| `getComponents` | Returns an array of all components registered with the application. |
| `Initialize` | Initializes the application and calls the load event. |
| `notifyScriptLoaded` | Indicates that the referenced script has been loaded. |
| `registerDisposableObject` | Registers an object that requires disposing with the application. |

## Sys Namespace 3-3

AJAX

### Component Class

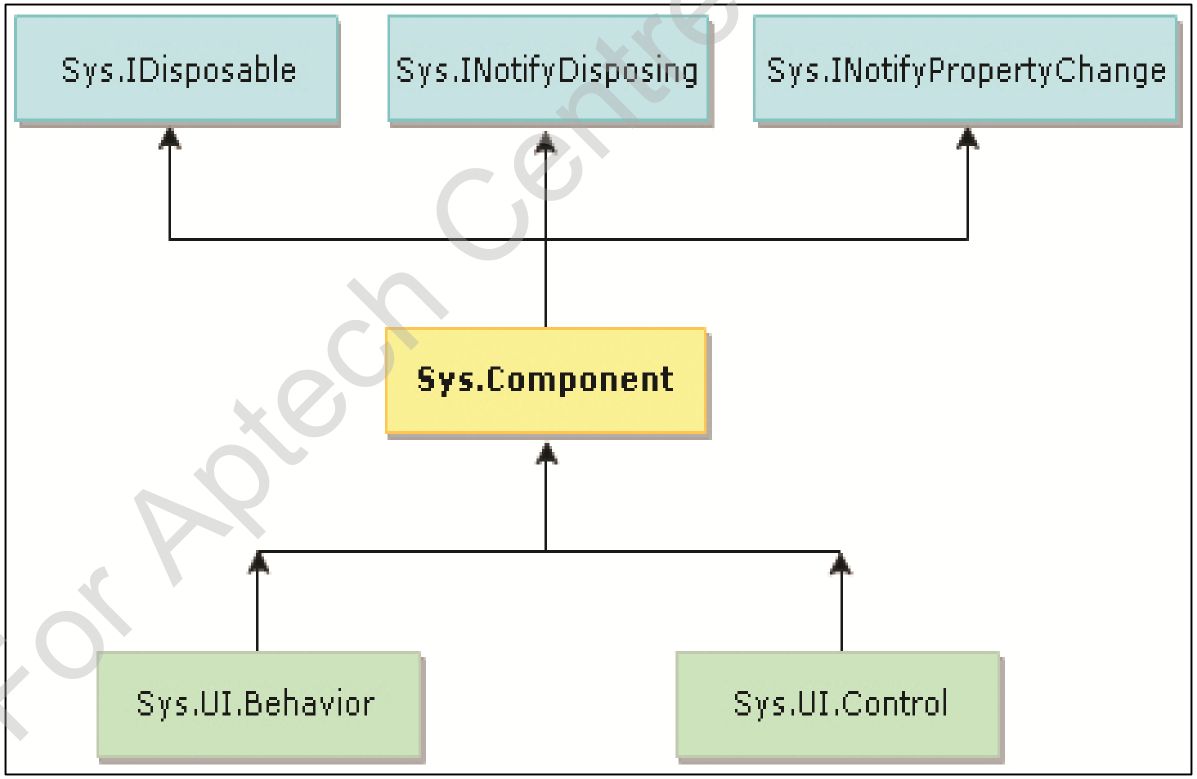
- Is the base class that is extended by all visual and non-visual client controls as shown in the following table.

| Name | Description |
|---|---|
| `beginUpdate` | Called by the `create` method to indicate that the process of setting properties of a component instance has begun. |
| `create` | Creates and initializes a component. |
| `dispose` | Removes the component from the application. |
| `endUpdate` | Called by the `create` method to indicate that the process of setting properties of a component instance has finished. |
| `initialize` | Initializes the component. |
| `raisePropertyChanged` | Raises the `propertyChanged` event of the current `Component` object. |
| `events` | Contains the references to all the event handlers that are mapped to the current component's events. |
| `id` | Specifies or retrieves the ID of the current `Component` object. |

## Custom Components

AJAX

- The `Sys.UI.Behavior` class is used to extend the behavior of an existing DOM element.
- The `Sys.UI.Control` class can be used to create visual controls that modify the original intent of a DOM element and extend an element's functionality.
- When you create a custom component, it automatically inherits the features from `Sys.Component` class as shown in the figure.

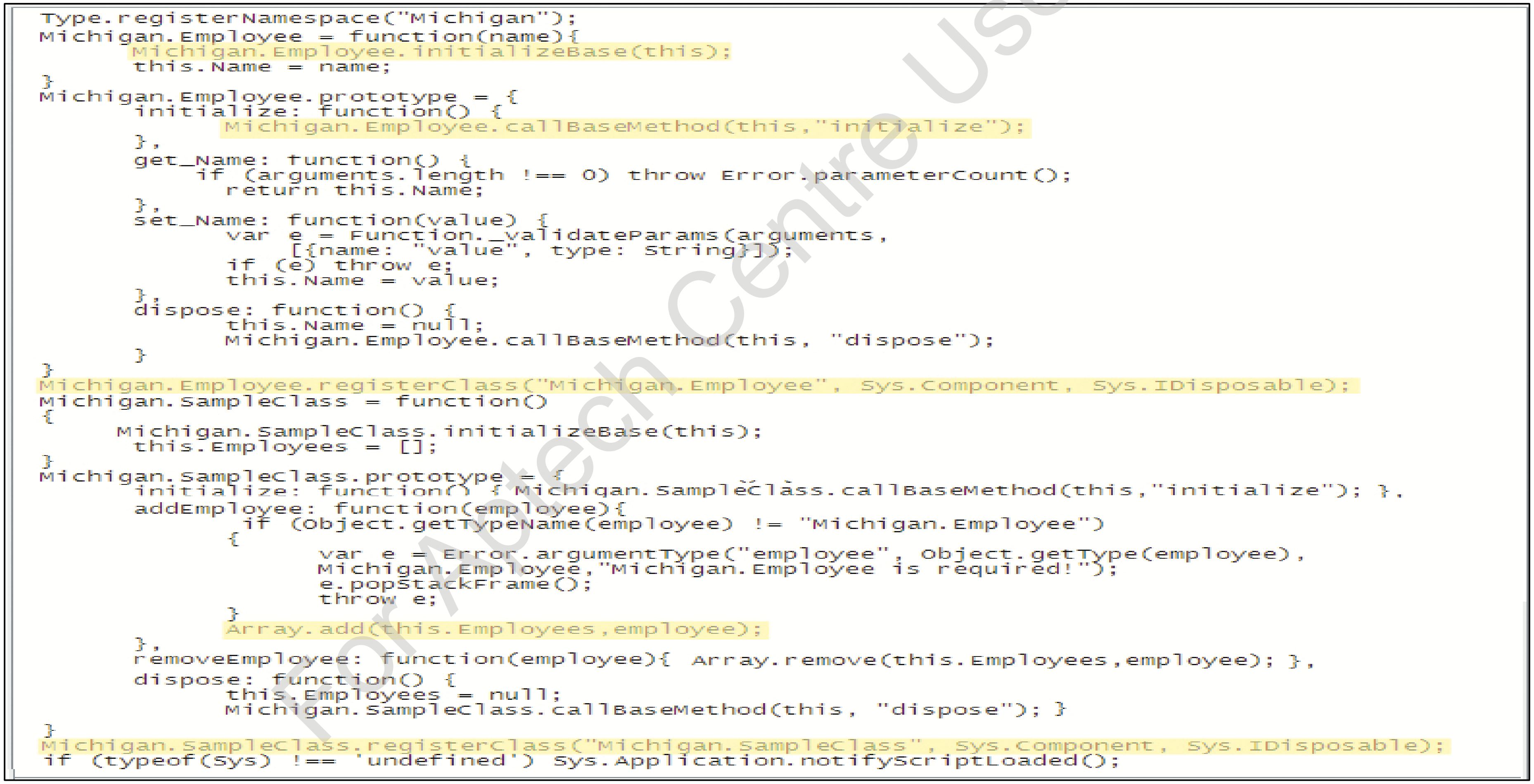


# Custom Components - Example 1-5

AJAX

## Step 1: Creating a Custom Component

- An example to create two classes named, `Employee` and `SampleClass` in `Michigan` namespace, is as shown in the figure.



```
Type.registerNamespace("Michigan");
Michigan.Employee = function(name){
      Michigan.Employee.initializeBase(this);
      this.Name = name;
}
Michigan.Employee.prototype = {
      initialize: function() {
            Michigan.Employee.callBaseMethod(this,"initialize");
      },
      get_Name: function() {
          if (arguments.length !== 0) throw Error.parameterCount();
            return this.Name;
      },
      set_Name: function(value) {
            var e = Function._validateParams(arguments,
                [{name: "value", type: String}]);
            if (e) throw e;
            this.Name = value;
      },
      dispose: function() {
            this.Name = null;
            Michigan.Employee.callBaseMethod(this, "dispose");
      }
}
Michigan.Employee.registerClass("Michigan.Employee", Sys.Component, Sys.IDisposable);
Michigan.SampleClass = function()
{
     Michigan.SampleClass.initializeBase(this);
      this.Employees = [];
}
Michigan.SampleClass.prototype = {
      initialize: function() { Michigan.SampleClass.callBaseMethod(this,"initialize"); },
      addEmployee: function(employee){
             if (Object.getTypeName(employee) != "Michigan.Employee")
            {
                  var e = Error.argumentType("employee", Object.getType(employee),
                  Michigan.Employee,"Michigan.Employee is required!");
                  e.popStackFrame();
                  throw e;
            }
            Array.add(this.Employees,employee);
      },
      removeEmployee: function(employee){ Array.remove(this.Employees,employee); },
      dispose: function() {
            this.Employees = null;
            Michigan.SampleClass.callBaseMethod(this, "dispose"); }
}
Michigan.SampleClass.registerClass("Michigan.SampleClass", Sys.Component, Sys.IDisposable);
if (typeof(Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();
```

## Custom Components - Example 2-5

AJAX

### Step 1: Creating a Custom Component

- Following code snippet creates a custom component.

**Code Snippet**

```
Michigan.Employee.initializeBase(this);
```

```
Michigan.Employee.registerClass("Michigan.Employee",
Sys.Component, Sys.IDisposable);
```

```
Array.add(this.Employees,employee);
```

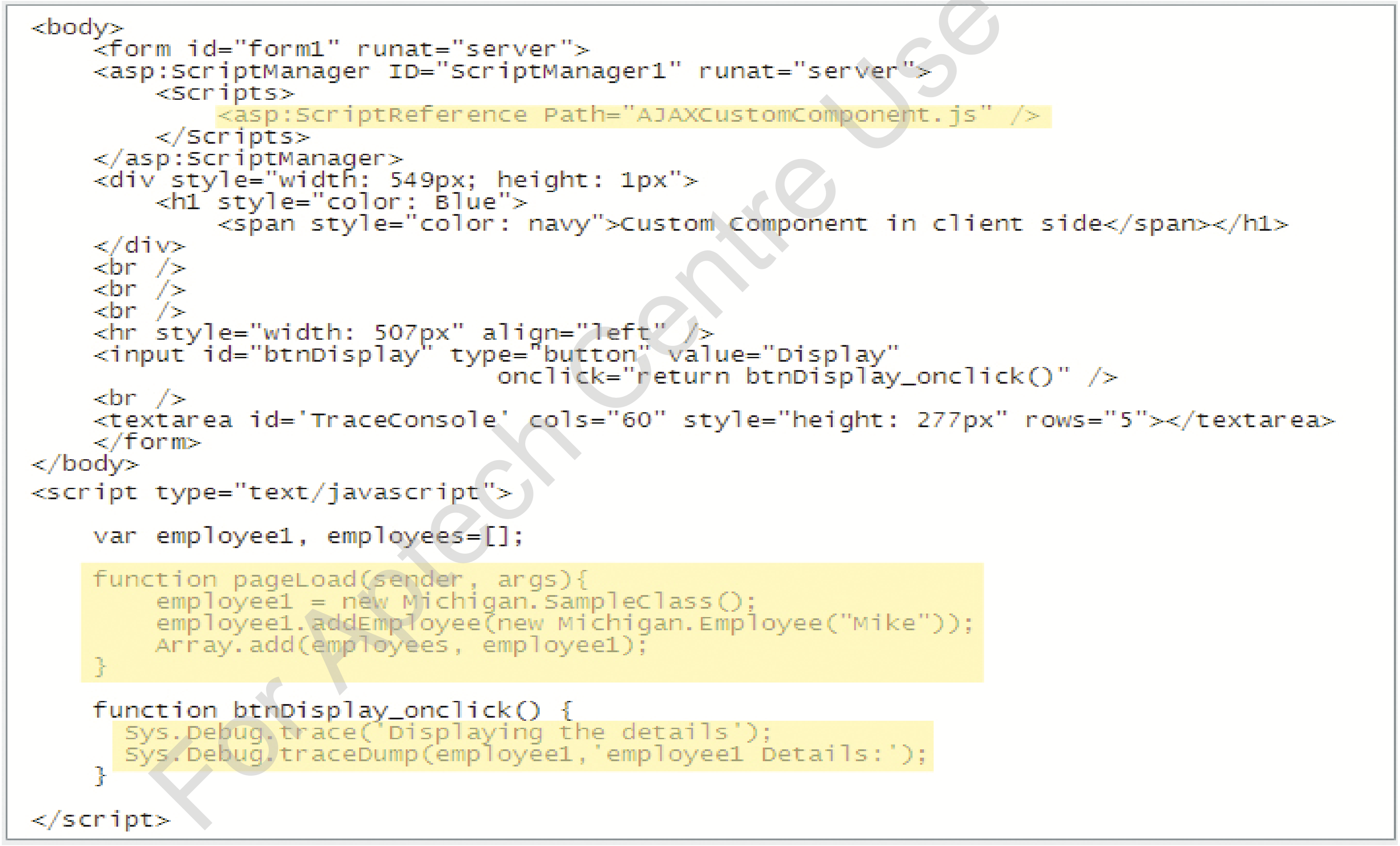
```
Michigan.SampleClass.registerClass("Michigan.SampleCl
ass", Sys.Component, Sys.IDisposable);
```

# Custom Components - Example 3-5

AJAX

## Step 2: Default.aspx

- The `Default.aspx` file as shown in the figure.



```
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Scripts>
            <asp:ScriptReference Path="AJAXCustomComponent.js" />
        </Scripts>
    </asp:ScriptManager>
    <div style="width: 549px; height: 1px">
        <h1 style="color: Blue">
            <span style="color: navy">Custom Component in client side</span></h1>
    </div>
    <br />
    <br />
    <br />
    <hr style="width: 507px" align="left" />
    <input id="btnDisplay" type="button" value="Display"
                          onclick="return btnDisplay_onclick()" />
    <br />
    <textarea id='TraceConsole' cols="60" style="height: 277px" rows="5"></textarea>
    </form>
</body>
<script type="text/javascript">

    var employee1, employees=[];

    function pageLoad(sender, args){
        employee1 = new Michigan.SampleClass();
        employee1.addEmployee(new Michigan.Employee("Mike"));
        Array.add(employees, employee1);
    }

    function btnDisplay_onclick() {
      Sys.Debug.trace('Displaying the details');
      Sys.Debug.traceDump(employee1,'employee1 Details:');
    }

</script>
```

## Custom Components - Example 4-5

AJAX

### Step 2: Default.aspx

- Following code snippet displays an example of `Default.aspx`.

**Code Snippet**

```
<asp:ScriptReference Path="AJAXCustomComponent.js" />
```

```
function pageLoad(sender, args){
        employee1 = new Michigan.SampleClass();
        employee1.addEmployee(new
Michigan.Employee("Mike"));
        Array.add(employees, employee1);
    }
```

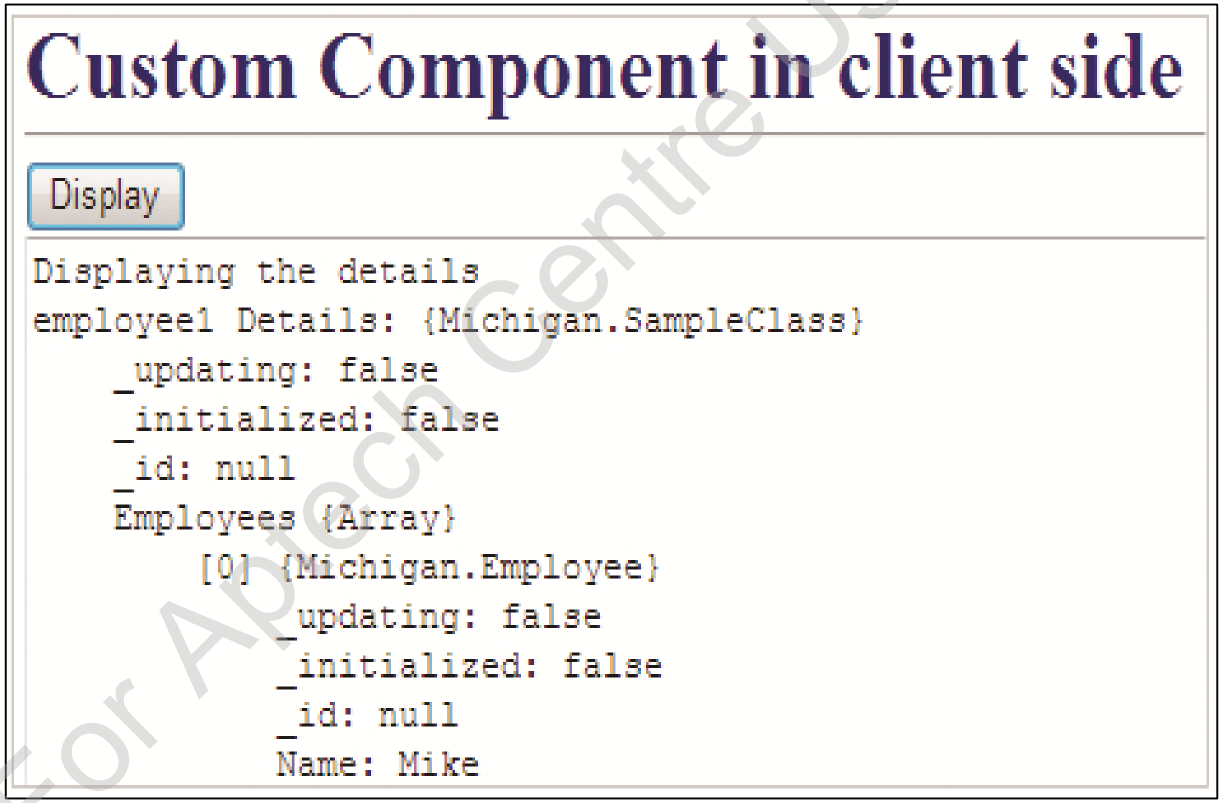
```
Sys.Debug.trace('Displaying the details');
Sys.Debug.traceDump(employee1,'employee1 Details:');
```

# Custom Components - Example 5-5

AJAX

## Output

- The output shows the various features of the `Component` class as well as the custom component details as shown in the figure.



```
Displaying the details
employee1 Details: {Michigan.SampleClass}
    _updating: false
    _initialized: false
    _id: null
    Employees {Array}
        [0] {Michigan.Employee}
            _updating: false
            _initialized: false
            _id: null
            Name: Mike
```

## Interfaces

- Is the root namespace in the Microsoft AJAX Library as shown in the following table.

| Name | Description |
|---|---|
| `IContainer` | Provides a common interface for all components that can contain other components. |
| `IDisposable` | Provides a common interface for closing or releasing resources held by instances of your registered Microsoft AJAX Library classes. |
| `INotifyDisposing` | Represents that the type that implements the interface provides disposing notifications. |
| `INotifyPropertyChange` | Defines the `propertyChanged` event. |

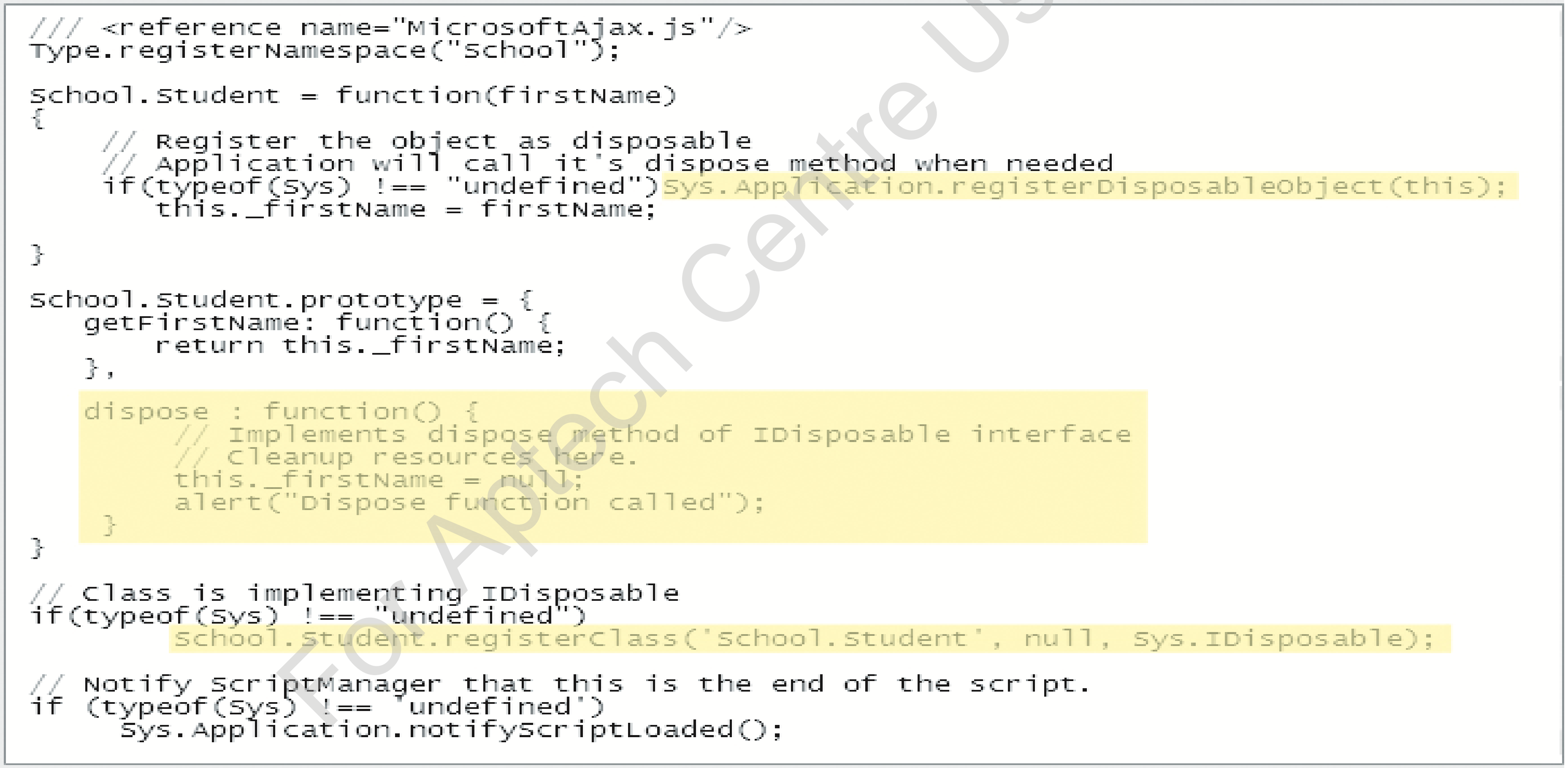
# Implementing IDisposable Interface 1-4

AJAX

- The `IDisposable` interface is used on the client-side to release the resources.

## Step 1: School.js code

- Following figure displays the `School.js` code.



```
/// <reference name="MicrosoftAjax.js"/>
Type.registerNamespace("School");

School.Student = function(firstName)
{
    // Register the object as disposable
    // Application will call it's dispose method when needed
    if(typeof(Sys) !== "undefined")Sys.Application.registerDisposableObject(this);
       this._firstName = firstName;

}

School.Student.prototype = {
   getFirstName: function() {
       return this._firstName;
   },

   dispose : function() {
        // Implements dispose method of IDisposable interface
        // Cleanup resources here.
        this._firstName = null;
        alert("Dispose function called");
    }
}

// Class is implementing IDisposable
if(typeof(Sys) !== "undefined")
        School.Student.registerClass('School.Student', null, Sys.IDisposable);

// Notify ScriptManager that this is the end of the script.
if (typeof(Sys) !== 'undefined')
     Sys.Application.notifyScriptLoaded();
```

## Implementing IDisposable Interface 2-4

AJAX

### Step 1: School.js code

- Following code snippet displays an example of `School.js` code.

**Code Snippet**

```
Sys.Application.registerDisposableObject(this)
```

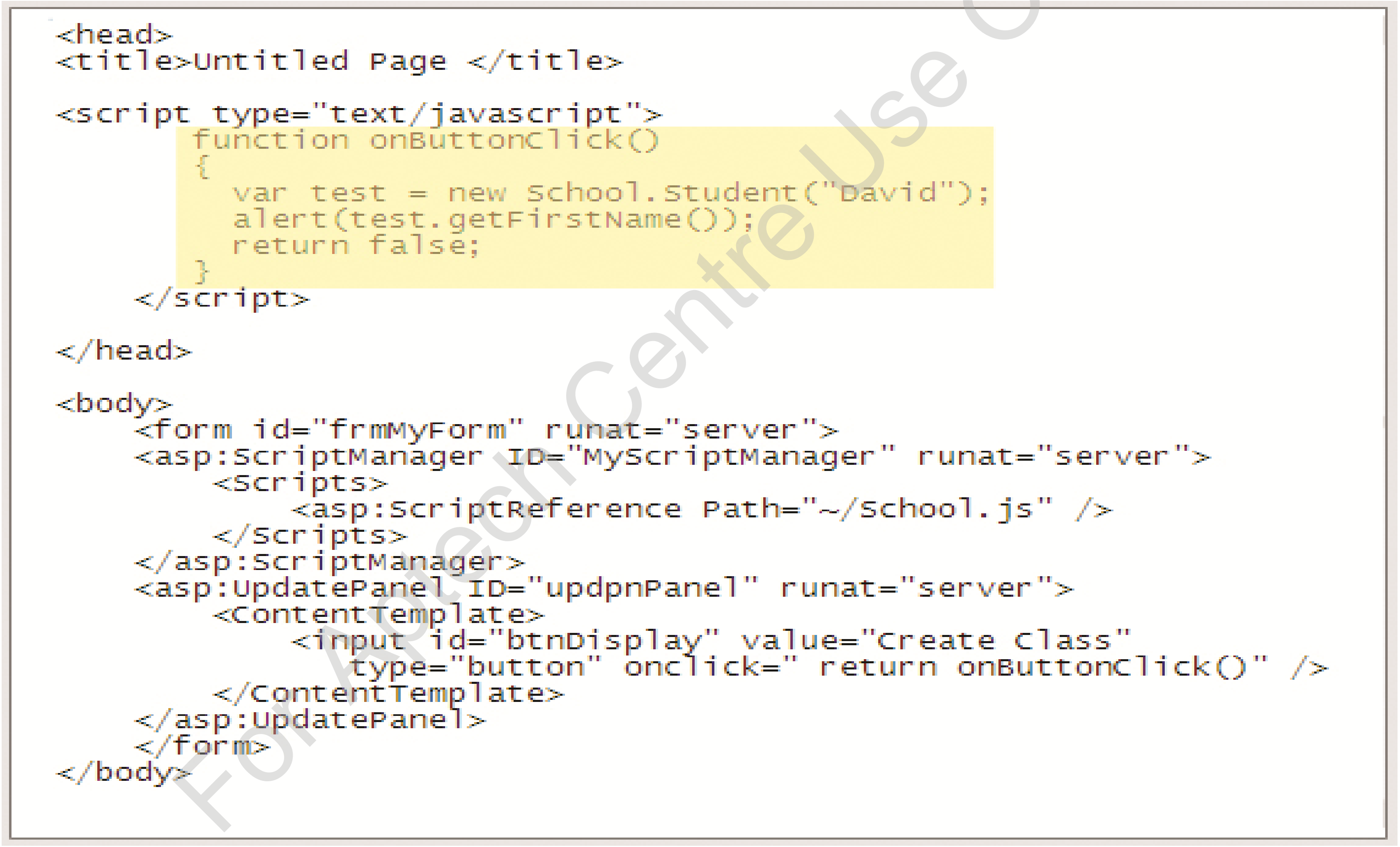
```
dispose : function() {
     // Implements dispose method of IDisposable
interface
        // Cleanup resources here.
        this._firstName = null;
        alert("Dispose function called");
    }
```

```
School.Student.registerClass('School.Student',null,
Sys.IDisposable);
```

## Implementing IDisposable Interface 3-4

AJAX

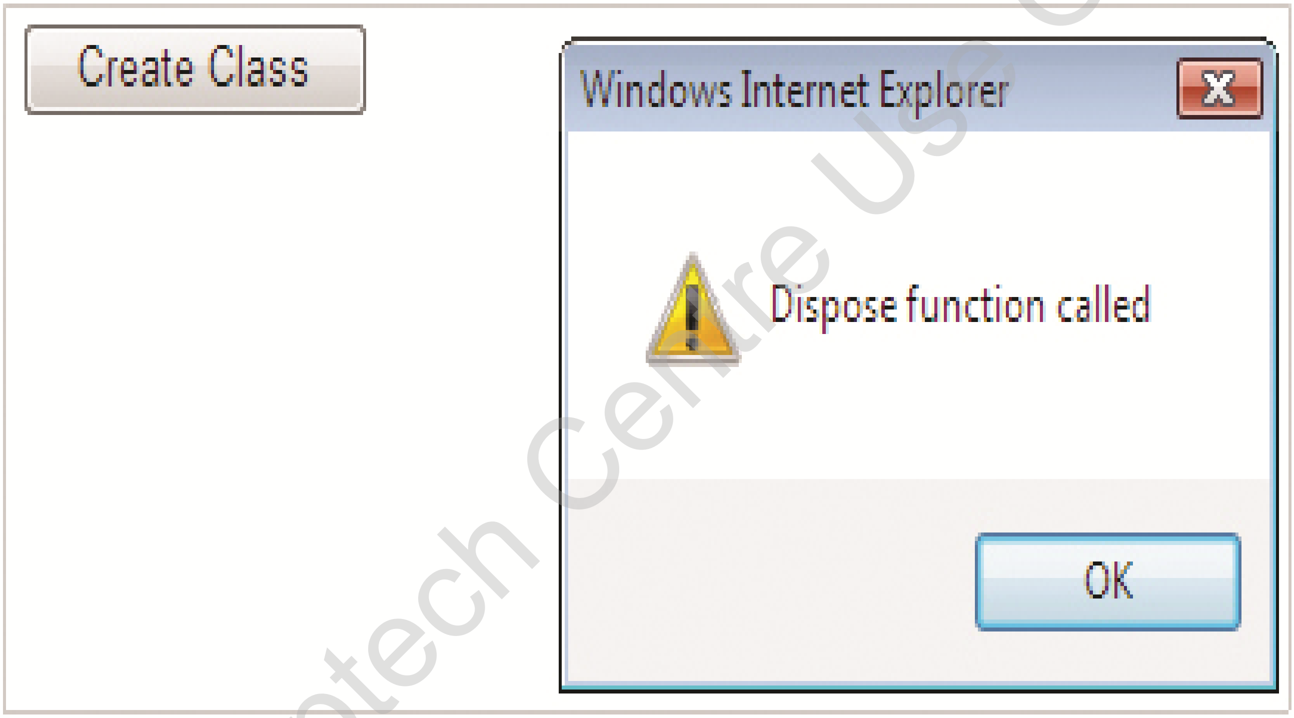
- Following figure displays the `Student` code.



```
<head>
<title>Untitled Page </title>

<script type="text/javascript">
       function onButtonClick()
       {
         var test = new School.Student("David");
         alert(test.getFirstName());
         return false;
       }
    </script>

</head>

<body>
    <form id="frmMyForm" runat="server">
    <asp:ScriptManager ID="MyScriptManager" runat="server">
        <Scripts>
            <asp:ScriptReference Path="~/School.js" />
        </Scripts>
    </asp:ScriptManager>
    <asp:UpdatePanel ID="updpnPanel" runat="server">
        <ContentTemplate>
            <input id="btnDisplay" value="Create Class"
               type="button" onclick=" return onButtonClick()" />
        </ContentTemplate>
    </asp:UpdatePanel>
    </form>
</body>
```

## Implementing `IDisposable` Interface 4-4

### Output



Dispose function called

Executes the dispose function when the user tries to close the Web page.

## Exceptions

AJAX

- The `Sys` namespace includes a number of exception types that you can use in any application.
- The exceptions are represented in the `Error` object that is raised by Microsoft AJAX Library Framework as shown in the table.

| Name | Description |
|---|---|
| `ArgumentException` | Raised when there is a mismatch in the arguments. |
| `ArgumentNullException` | Raised when one of the argument value is `null`. |
| `ArgumentUndefinedException` | Raised when one of the argument is undefined. |
| `InvalidOperationException` | Raised when a call to a method fails. |
| `ScriptLoadFailedException` | Raised when a script is not loaded successfully. |

## Sys.UI Namespace – Classes

AJAX

- The `Sys.UI` namespace provides an environment for creating client visual controls.
- The namespace provides classes for controlling DOM elements and events in the browser as shown in the table.

| Name | Description |
|---|---|
| `Behavior` | Provides a base class for all ASP.NET AJAX client behaviors. |
| `Bounds` | Constructs an object containing a number of properties for a specific position such as position, width, and height. |
| `Control` | Provides a base class for all ASP.NET AJAX client controls. |
| `DomElement` | Represents the main class for handling client-side controls in the browser DOM. |
| `DomEvent` | Represents the main class for handling the client-side events in the browser. |
| `Point` | Represents an object containing the integer coordinates of a position. |

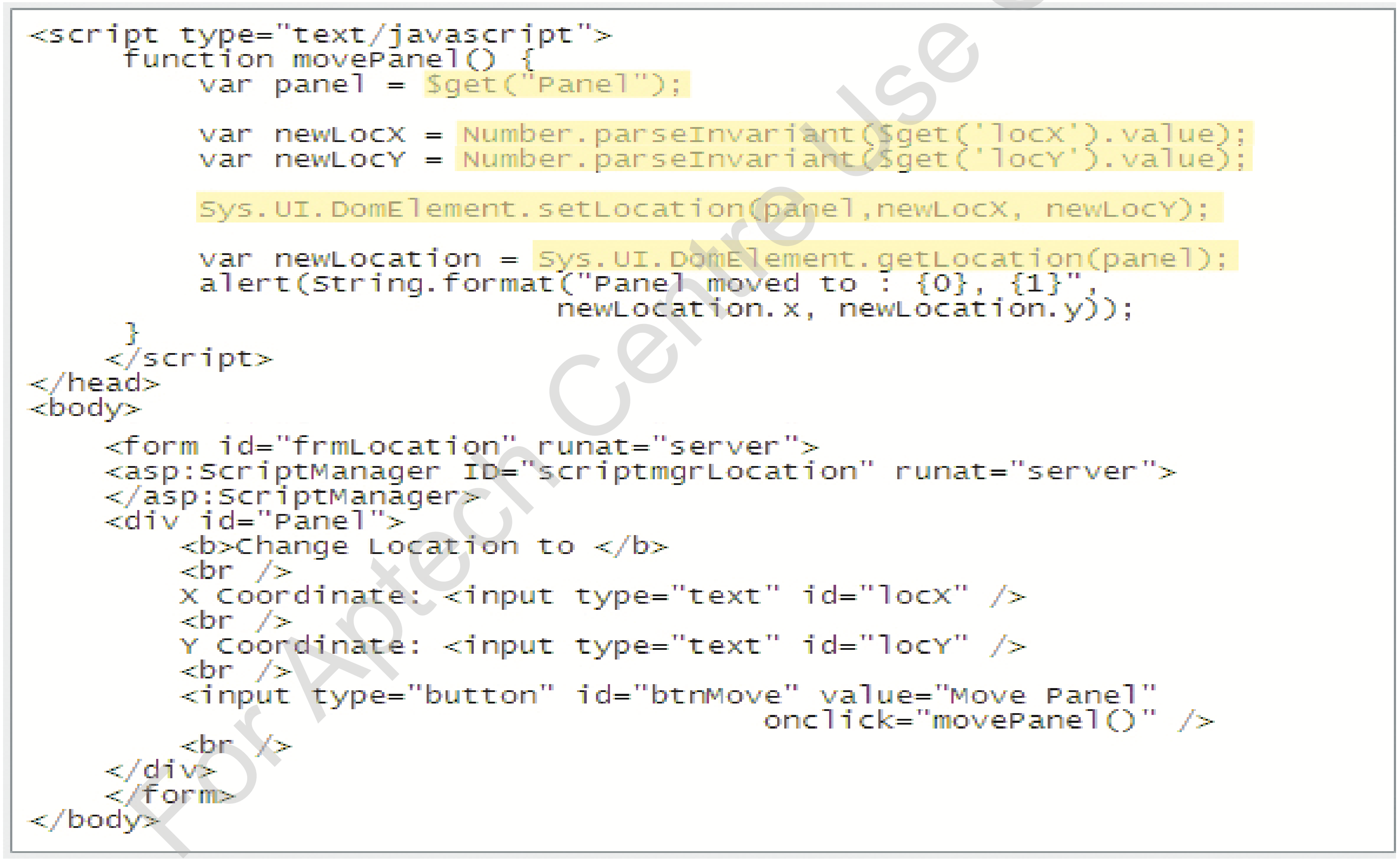
# DomElement Class

- Provides a better, consistent way of handling DOM elements in the browser as shown in the following table.

| Name | Description |
|---|---|
| `addCssClass` | Adds a CSS class to a DOM element. |
| `getElementById` | Returns a DOM element by the id element. (`$get` is the shortcut to this method). |
| `getLocation` | Returns the absolute position of a DOM element. |
| `removeCssClass` | Removes a CSS class from a DOM element. |
| `setLocation` | Sets the position of a DOM element. |

## DomElement Class - Example 1-4

- The members of the `Sys.UI.DomElement` class are used to change the location of controls in Web pages as shown in the figure.



```
<script type="text/javascript">
     function movePanel() {
         var panel = $get("Panel");

         var newLocX = Number.parseInvariant($get('locX').value);
         var newLocY = Number.parseInvariant($get('locY').value);

         Sys.UI.DomElement.setLocation(panel,newLocX, newLocY);

         var newLocation = Sys.UI.DomElement.getLocation(panel);
         alert(String.format("Panel moved to : {0}, {1}",
                             newLocation.x, newLocation.y));
     }
    </script>
</head>
<body>
    <form id="frmLocation" runat="server">
    <asp:ScriptManager ID="scriptmgrLocation" runat="server">
    </asp:ScriptManager>
    <div id="Panel">
        <b>Change Location to </b>
        <br />
        X Coordinate: <input type="text" id="locX" />
        <br />
        Y Coordinate: <input type="text" id="locY" />
        <br />
        <input type="button" id="btnMove" value="Move Panel"
                                          onclick="movePanel()" />
        <br />
    </div>
    </form>
</body>
```

## DomElement Class - Example 2-4

AJAX

- Following code snippet shows the `DomElement` code.

**Code Snippet**

```
$get("Panel");
```

```
Number.parseInvariant($get('locX').value);
```

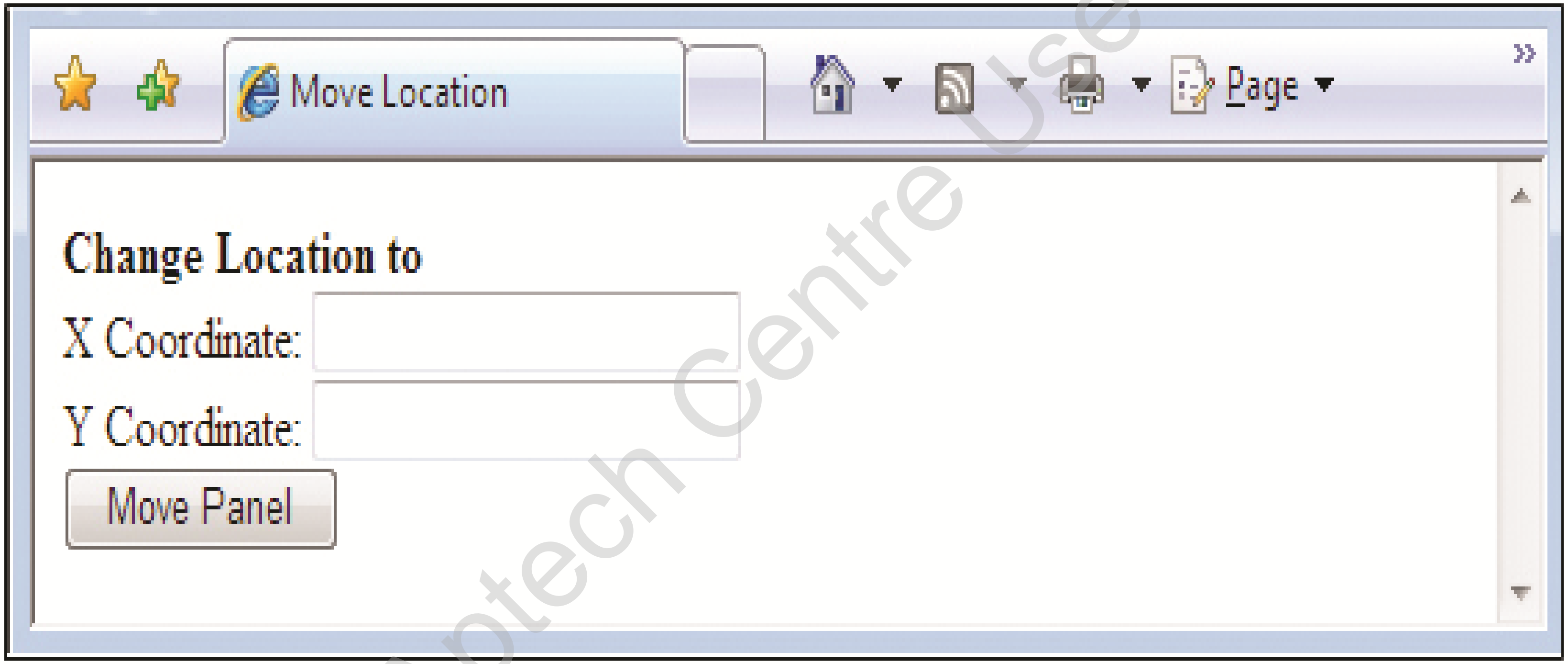
```
Number.parseInvariant($get('locY').value);
```

```
Sys.UI.DomElement.setLocation(panel, newLocX, newLocY);
```

```
Sys.UI.DomElement.getLocation(panel);
```

## DomElement Class - Example 3-4

Following figure displays the initial location.



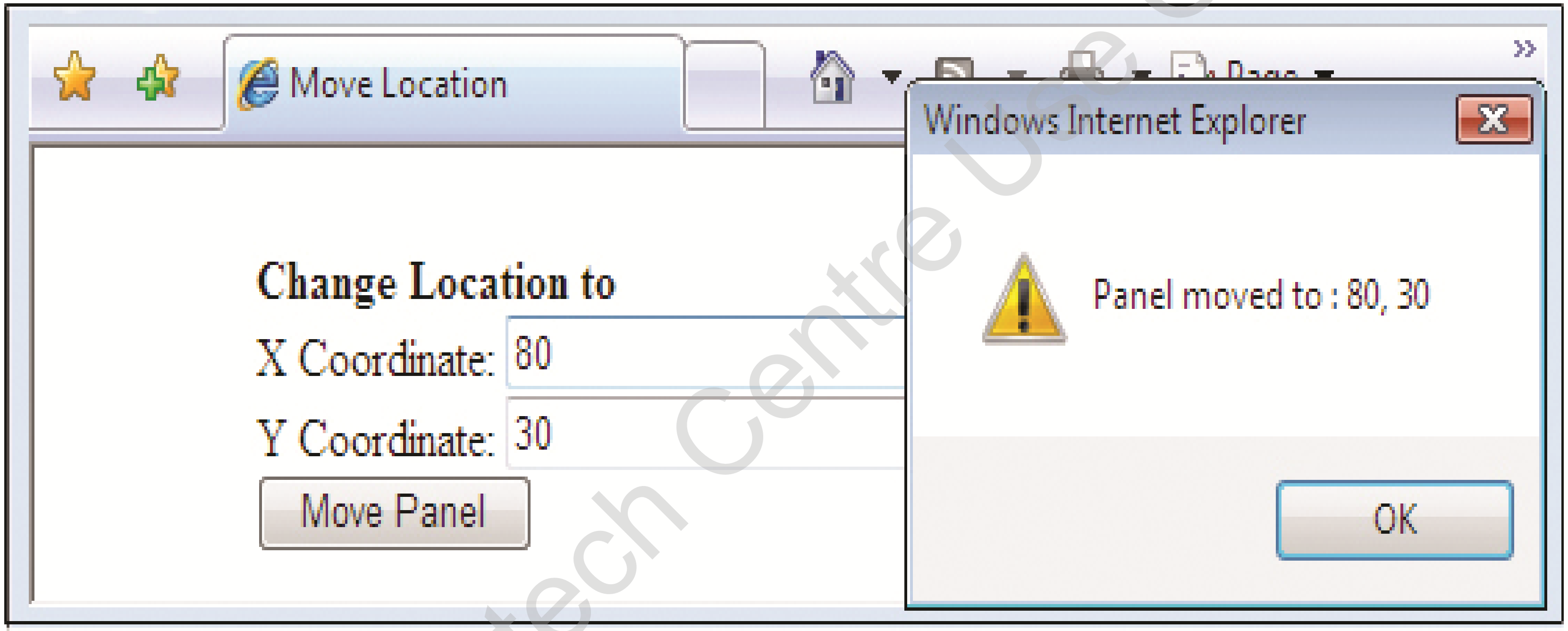
Change Location to

X Coordinate:

Y Coordinate:

## DomElement Class - Example 4-4

**Following figure displays the new location in the output.**



Change Location to

X Coordinate: 80

Y Coordinate: 30

Panel moved to : 80, 30

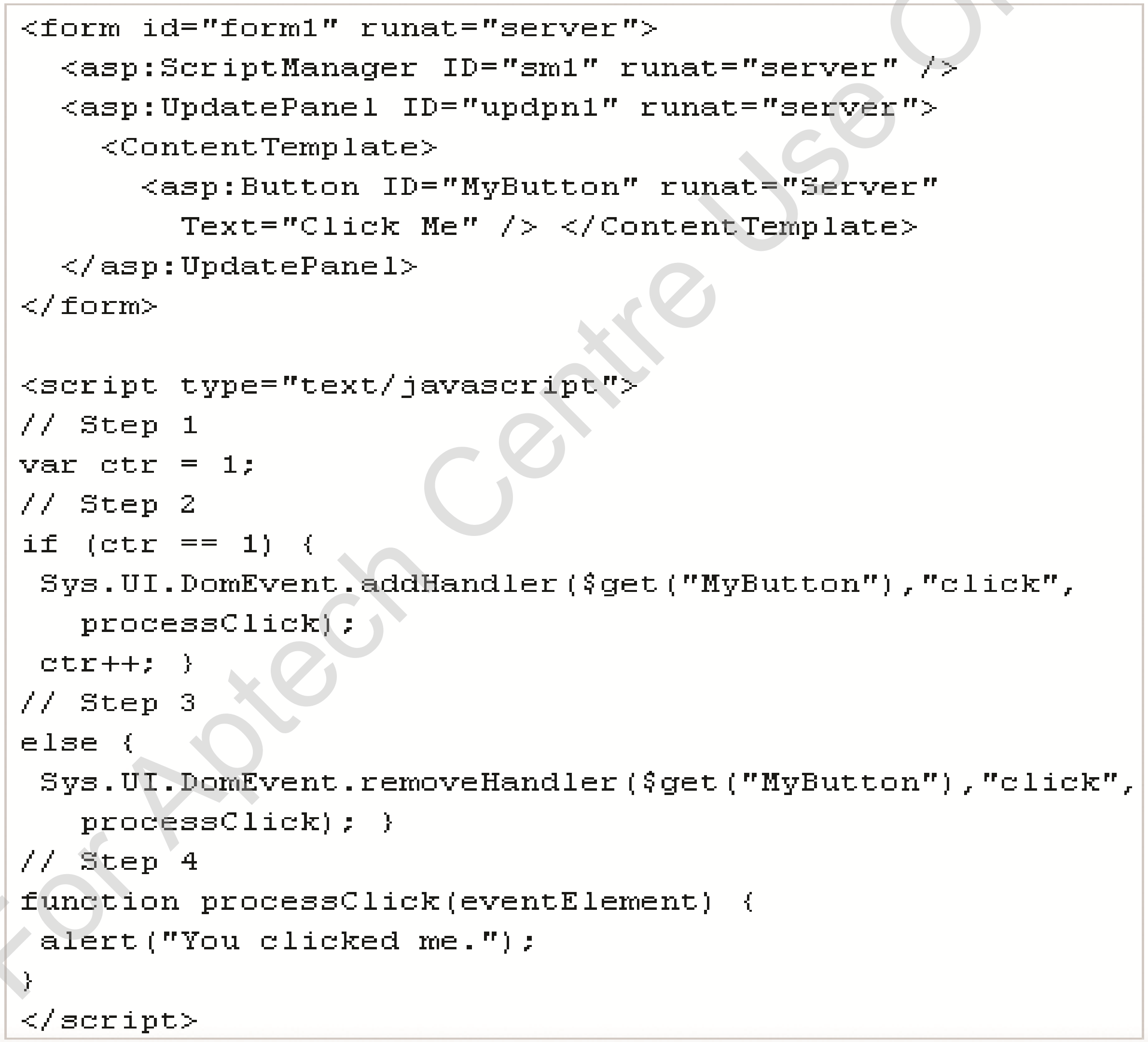
## DomEvent Class

- The `DomEvent` class in `Sys.UI` namespace represents an event data object that DOM event handlers receive as an argument as shown in the table.

| Name | Description |
|---|---|
| `addHandler` | Add a DOM event handler to an element. |
| `preventDefault` | Prevents the default event action from being raised. |
| `removeHandler` | Removes an event handler from the specified element. |
| `stopPropagation` | Prevents an event from being propagated to parent elements. |

# Event Handlers Example 1-2

- Following figure displays the event handler.



```
<form id="form1" runat="server">
  <asp:ScriptManager ID="sm1" runat="server" />
  <asp:UpdatePanel ID="updpn1" runat="server">
    <ContentTemplate>
      <asp:Button ID="MyButton" runat="Server"
        Text="Click Me" /> </ContentTemplate>
  </asp:UpdatePanel>
</form>

<script type="text/javascript">
// Step 1
var ctr = 1;
// Step 2
if (ctr == 1) {
 Sys.UI.DomEvent.addHandler($get("MyButton"),"click",
   processClick);
 ctr++; }
// Step 3
else {
 Sys.UI.DomEvent.removeHandler($get("MyButton"),"click",
   processClick); }
// Step 4
function processClick(eventElement) {
 alert("You clicked me.");
}
</script>
```

## Event Handlers Example 2-2

AJAX

Step 1

- A variable, `ctr` is created with initial value as 1.

Step 2

- An event named, `click`, is attached to the button `MyButton` by `addHandler` method.

Step 3

- The event from the element, `MyButton`, is detached by using the `removeHandler` method.

Step 4

- The code for the `click` event is written.

## DomEvent Class – Fields

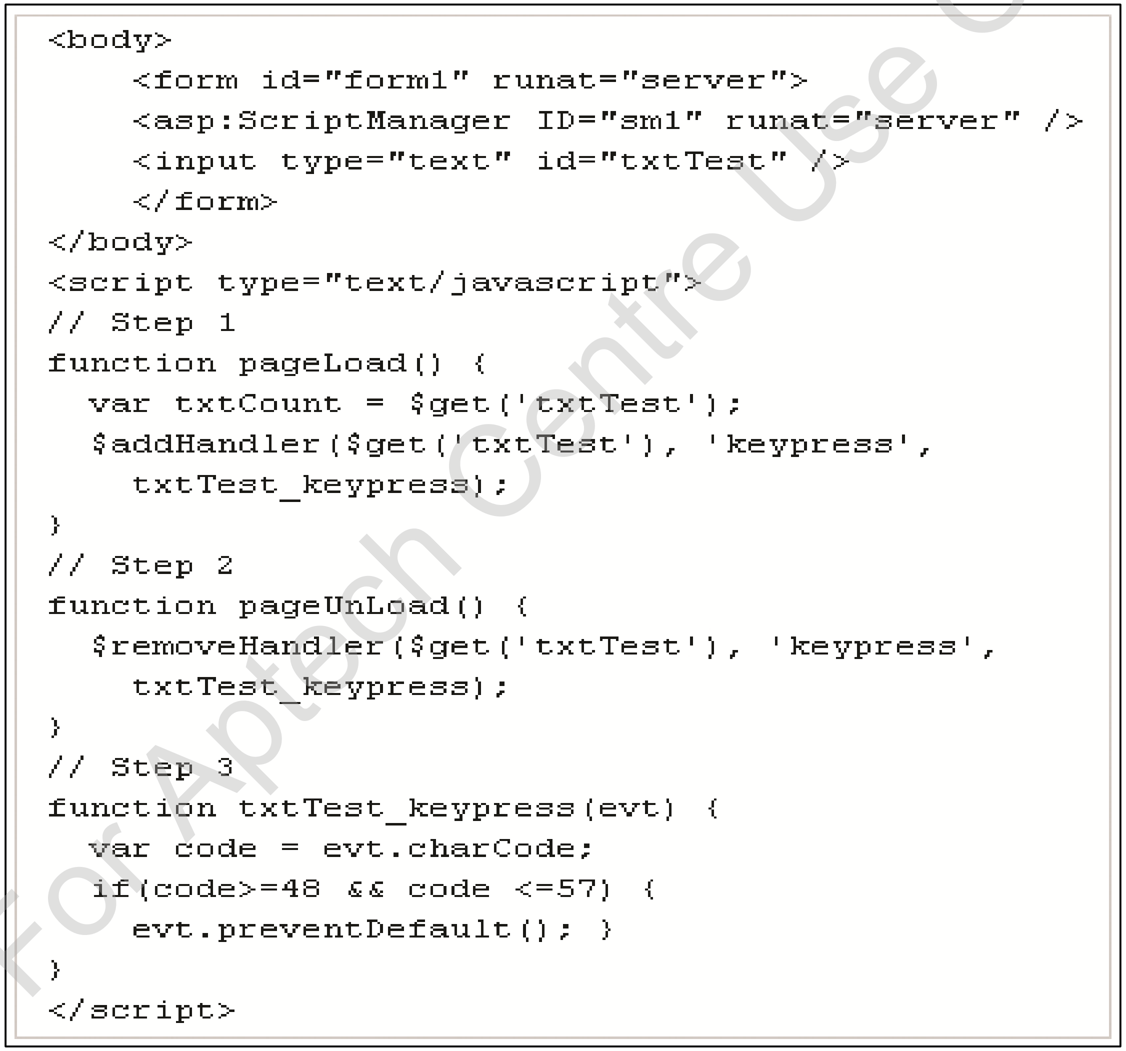
- The fields in the DomEvent class are used to trap the key codes, find the screen coordinates and trap special keys as shown in the table.

| Name | Description |
|---|---|
| button | Returns one of the Sys.UI.MouseButton enumeration values. The values are leftButton, middleButton and rightButton. |
| charCode | Returns an integer value that represents the key that is pressed. |
| clientX | Returns the x-coordinate value of the mouse pointer position relative to the client area of the browser window. |
| clientY | Returns the y-coordinate value of the mouse pointer position relative to the client area of the browser window. |
| screenX | Returns the x-coordinate value of the mouse pointer position relative to the user's screen. |
| screenY | Returns the y-coordinate value of the mouse pointer position relative to the user's screen. |
| target | Returns the object that raised the event. |
| type | Returns the name of the event that is raised. |

## Key Codes Example 1-2

AJAX

- Following figure displays the key codes example.



```
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="sm1" runat="server" />
    <input type="text" id="txtTest" />
    </form>
</body>
<script type="text/javascript">
// Step 1
function pageLoad() {
  var txtCount = $get('txtTest');
  $addHandler($get('txtTest'), 'keypress',
    txtTest_keypress);
}
// Step 2
function pageUnLoad() {
  $removeHandler($get('txtTest'), 'keypress',
    txtTest_keypress);
}
// Step 3
function txtTest_keypress(evt) {
  var code = evt.charCode;
  if(code>=48 && code <=57) {
    evt.preventDefault(); }
}
</script>
```

## Key Codes Example 2-2

Step 1

- Adds the event handler, `txtTest_keypress`, for the event `keypress`.

Step 2

- Removes the event from the `txtTest` element.

Step 3

- Defines the JavaScript function `txtTest_keypress`, the event handler for the text box element.

## Summary

- Tools to create client applications that are not dependent on the browser are provided by Microsoft AJAX Client Library.
- Portions of a page can be updated using the UpdatePanel control, without the whole user interface being refreshed.
- Object-oriented features are made available for JavaScript objects and AJAX-enabled ASP.NET applications by Microsoft AJAX Library’s type system and extensions.
- Functionalities of JavaScript base types can be extended using the members of the Microsoft AJAX Library type extensions.
- The built-in JavaScript Error objects have added functionality in the form of Error type extension. The exception details are generated by the Error object.
- The base class extended by all visual and non-visual client controls, is the Component class.
- Client visual controls are created using the Sys.UI namespace.