|  |  |
| --- | --- |
| Parent Object   * Child Object * Child Property   Child object being accessed through a Collection []   * Event * Method() | Document   * CreateDocumentFragment() * CreateElement(‘el\_nam’) * CreateTextNode(‘txt\_el’)   \*Access through DOM   * body * documentElement * getElementById(‘ID’) * getElementByTagName(‘Tag\_name’) * createEvent(event module) * createEventObject(template obj) |

**Node**

|  |  |
| --- | --- |
| Node   * childNodes[] * parentNode * firstChild * getElementsByTagName(‘Tag’) * lastChild * nextSibling * offsetParent * previousSibling   Attaching, Copying, Removing   * appendChild(nodeRef) * cloneNode(bool:copy children or not) * innerHTML * insertBefore(bf\_node,c\_node) * insertAfter(af\_node,c\_node) * removeChild(ref) * replaceChild(rp\_child,c\_child) * splitText   Size & Position   * clientHeight * clientWidth * offsetHeight * offsetLeft * offsetParent * offsetTop | Node\_info   * data * hasChildNodes() * id * nodeName * nodeType * nodeValue * specified * tagName * title   Attributes   * attributes[] * getAttribute(‘name’) * removeAttribute(‘name’) * setAttribute(‘name’,’value’)   Style   * className   currentStyle.styleName  runtimeStyle.styleName  style.styleName  Size & Position   * offsetWidth * scrollHeight * scrollLeft * scrollTop * scrollWidth |

Table

|  |  |
| --- | --- |
| Table   * caption * cells[] * rows[] * tBody[] * tfoot * thead | Attaching, Copying, Removing   * createCaption() * createTfoot()   Node info   * cellIndex * rowIndex * sectionRowIndex |

Event

|  |  |
| --- | --- |
| Node   * addEventListener(event,function,phase) * attachEvent(event,function) * detachEvent(event,function) * removeEventListener(event,function,phase)   document   * createEvent(event module) * createEventObject(template obj)   Firing Events   * dispatchEvent(event obj) * fireEvent(‘ontype’,event obj)   event object   * initEvent(‘type’,bubbles,cancelable) * initKeyEvent(‘type’,bubbles,cancelable,window,ctrlKey,altKey,shiftKey,keycode,charcode) * initMouseEvent(‘type’, bubbles,cancelable,window,detail,screenX,screenY,clientX,client,Y ctrlKey,altKey,shiftKey,mouseKey,Target) * initMutationEvent(‘type’,bubbles,cancelable,relatedNode,preValue,New value,attr Name,attr change) * initUiEvent(‘type’,bubbles,cancelable,window,detail) | Event object   * attrChange * attrName * bubbles * cancelable * cancelBubble * charCode * currentTarget * detail * eventPhase * fromElement * metakey * newValue * preventDefault() * prevValue * relatedNode * relatedTarget * returnValue * stopPropagation() * toElement |

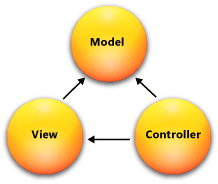
**Style sheets**

|  |  |
| --- | --- |
| * addRule(selection,style) * cssRules[] * cssRules[] * cssText * deleteRule(index) * encoding * href * insertRule(rule,index) * media | * parentRule * style   + - * + length         + nameofStyle       * styleSheet       * type * deleteRule(index) * ownerNode * owningElement * title * type |

|  |  |
| --- | --- |
| **Model-View-Controller** | **Model-View-Presenter** |
|  |  |
| View  Receives model elements from controller. Displays and interacts with element. | View  Receives data from presenter, notifies presenter of interactions. |
| Controller  Supervises view/model, performs actions in response to events/changes. | Presenter  Delegates events to model. Re-maps model outgoing events into state changes. |

ActionResult Return Type

Most action methods return an instance of a class that derives from ActionResult. The ActionResult class is the base for all action results. However, there are different action result types, depending on the task that the action method is performing. For example, the most common action is to call the View method. The View method returns an instance of the ViewResult class, which is derived from ActionResult.



**Area**

The MVC pattern separates the model (data) logic of an application from its presentation logic and business logic. In ASP.NET MVC, this logical separation is also implemented physically in the project structure, where controllers and views are kept in folders that use naming conventions to define relationships. This structure supports the needs of most Web applications.

However, some applications can have a large number of controllers, and each controller can be associated with several views. For these types of applications, the default ASP.NET MVC project structure can become unwieldy.

To accommodate large projects, ASP.NET MVC lets you partition Web applications into smaller units that are referred to as areas. Areas provide a way to separate a large MVC Web application into smaller functional groupings. An area is effectively an MVC structure inside an application. An application could contain several MVC structures (areas).

* Main. This is entry point to the Web application. This area includes the landing page and a log-in *feature.*
* Blog. This area is used to display blog posts and to search the archive.
* Dashboard. This area is used to create and edit blog posts.

Function DataSetToJSON(ds As DataSet) As String

Dim dict As New Dictionary(Of String, Object)

For Each dt As DataTable In ds.Tables

Dim arr(dt.Rows.Count) As Object

For i As Integer = 0 To dt.Rows.Count - 1

arr(i) = dt.Rows(i).ItemArray

Next

dict.Add(dt.TableName, arr)

Next

Dim json As New JavaScriptSerializer

Return json.Serialize(dict)

End Function

