

## Replacing the Explorer context menu with a custom context menu

### Replacing the Internet Explorer context menu with a custom built context menu

This case study describes how to replace the default context menu shipped with the explorer, with a context menu that allows the programmer to set its pop-up items for each HTML tag, or HTML element with a name attribute and menu items that will always be displayed. (such as Add, Back&hellip;).

### Adding items to Explorer context (right-click) menu easily &ndash; How to add them ?

Add items to Windows Explorer Shell context menu easily with Windows Explorer Shell Context Menu. This powerful .Net component for custom items appending to Explorer context menu will add all your application entries to Windows Explorer Shell context menu. This .Net component with full C# , C++ and VB.NET support include detailed C# / VB.NET samples, tutorials and support all you may need :

- Add all your items to Windows Explorer Shell context menu to be shown on any Windows operating system (all operating systems are supported &ndash; Windows XP, Vista, Windows x64 , etc.)
- Add all your items to Windows Explorer Shell context menu to be shown in any way - with custom caption and icon, as separator or sub-menu
- Add items to Windows Explorer Shell context menu to be shown for all files or shown only for files of particular type (for example, only for .PDF .TXT , .MP3,.WMA,.AAC , .MPG media files)
- Add your program items to Windows Explorer Shell context menu, sub-menus, sub-sub-menus, sub-menus of unlimited depth and add to Explorer context menu entries of all types

Windows Explorer Shell Context Menu - is a .Net framework component that support all you may need to add all your items to the Windows Explorer Shell context menu - in a fast and easy way. Add your application items to Explorer context menu right now &ndash; fast and exactly as you prefer :

### Replacing the Internet Explorer context menu with a custom built context menu

Here is described how to add custom items to Internet Explorer and to Windows Explorer Shell context menu, but context menu customization method works only for Windows 95 / Windows 98 (not on XP, Vista, x64 - 64-bit Windows), to add items to Windows Explorer Shell context menu you should use, according to Microsoft guidelines, appropriate .Net component - Windows Explorer Shell Context Menu. The sample window shows the menu implementation. The Add and Update items are fixed for every table in the application (. elements in the table may be added or updated at any time). The E-mail icon is set so that the user may email customers. The items under the dashed line are global items that are available to the user by right-clicking.

The programmer can also set the location of the event (client or server) that will appear when the user selects one of the menu items. This suggested mechanism enhances the default context menu by enabling the user to set a context menu with type of tags and specific instances of tags with name attributes. Most of the Man Machine Interface (MMI) specialists agree that enabling sensitive right click applications helps the end user navigate in complex applications (see the links). Usually complex applications in the web area are Intranet applications. Data from many enterprise sources and types are show on HTML pages in order to help the enterprise reach its objectives (in most cases applications that were previously on Mainframe, AS400 etc.). In such applications the user requires a pop-up menu that meets the functionality of the data that he is working with. For example every table deals with a set of data that the user can update, add new and delete. If a specific table shows a list of customer we may want to add new menu item that will enable us to send e-mail to customers. The menu system is based on an XML file that defines menu items for html types and item names and the event location, a base Form class that encapsulates the underlying code, a web service and a client side script.

It all started when a lady (who is responsible for the man machine interface (MMI) at one of my clients) returned from the last CHI (Conference on Human Factors in Computing Systems) summit with new user interface requirements. One of these requirements was for a dynamic menu on the browser, similar to the one that was shipped with Explorer but with new requirements. She wanted the menu items to be dynamic by right-clicking on the HTML element tag and name. After

all, the context menu of a list of workers isn't the same as the context menu for a list of shipments.

There are some issues that we need to address. The first is how and where to keep the data that the programmer defines in order to control the menu items. We want to enable the programmer to be able to set the menu items in one place to prevent updating the data in many places. But if we need to call on the server each and every time in order to know which items the programmer has set for display, we will have performance problems and stress on the site. Secondly, we need a way to add the menu dynamically without sending the whole page back to the server. There is no existing context menu that can be used to set the items dynamically or determine the location of the menu items events, so we need to use DHTML to build one. Third and finally, we need to handle the event appearing on the server side and at the client side.

In order to solve these problems use several techniques. We will create an XML schema that defines the way the programmer sets the menu items to HTML types and elements names. To solve the performance issue, we will cache the XML the first time we use it. Then we will write script code at the client side in order to seize the events and handle them. Further, we need a cache menu data at the client side to prevent round trips to the server. A web service will obtain the HTML tag and name, and return HTML that contains the menu elements that match the given HTML type and name.

Using web service implies that there will be performance problems, and some issues need to be considered. The programmer creates an XML file on the server that holds all the instructions required in order to build the context menu for each element on the page. There are three ways that we can implement this.

- The XML file can be downloaded each and every time that we download a page. Then using DHTML and scripting we can build the menu. There are some disadvantages to this method.. The first refers to the size of the XML page. If the XML file is large it will affect the page download time. Secondly, there is possibility that the user will not use most of the menus that were download.
- Call up the server every time we need to build a menu. At server level, we define the menu that meets the right click item definitions. This implementation prevents us downloading items that will not be used. The disadvantage of this approach is the effect on performance . The need to employ the server every time we will place stress on it and also effect the network. Even if we use the web service as another line to the server which will retrieve only the menu data, we will still need to go to the server every time we right click on an item.
- This is a combination of the two methods above. The first time the user right clicks on an item we use web service (as an asynchrony connection to the server) to retrieve the menu data and to cache it at the client. In this way, when the user right clicks on an item that is already cached we will use the cached data and not the server. Using this method, only the data needed to create the menu is downloaded, and there is no download of unnecessary data and reduces dramatically the roundtrip time to the server.

The final issue for consideration concerns the application of the event on the server. In order to encapsulate most of the client side script we will create a super class page.

## The Problem and Solution

The problems usually appear in enterprises with many in-house web applications existing on the enterprise servers. Those applications are actually seen to be a large single application that handles the enterprise legacy. The user of such an application moves from sub-application to sub-application without being aware that each is different application built by different programmers. This problem can be solved by maintaining uniformity of user interface design. This uniformity includes colors, fonts, locations of controls on the screen according to their functionality (general buttons at the top, screen specific buttons at the bottom, navigate tree view at the left and so on). This uniformity and other regulations enable the user to operate all the screens of applications in the same way. One of the regulations is a context menu with common items for all right click objects and a set of items that matches certain HTML tag and items that are unique for a certain HTML object. You probably understand that the goal of this article is to enable the organization programmer to define a mechanism that will enable the creation of a dynamic menu for all the enterprise applications. Writing such a mechanism raises problems that require attention:

- How to maintain the definition of the menus tags and element names? Where to keep the menu items that are common to all the applications and the menu items that are used by special applications or pages. We want to maintain definitions that will be effective for every site that we build, with the additional ability to overload certain menu items in a specific site. We also want to define the menu items for HTML tags and to overload them to a specific HTML object.

- When the user right clicks on an HTML element we obviously know which specific control the user selected. Possibilities exist to retrieve the data from the server the right menu can be built. We can use server events to build the menu on the server , but this implementation will result in re-creation of the page or involve transferring large amounts of data via the page view-state. An alternative is to retrieve all the menu data at the client and then recall it when the user right clicks on object. A further alternative is to call the server for the data at the time needed and then cache the data at

the client. The last solution is the probably the optimum. In order to use it, a method for calling up the server from the page is required, so that the menu data can be retrieved and converted into a visual menu that will be available to the user will on right clicking.

- Our last concern is to create the mechanism that will enable uploading events at either the client or the server side, according to the programmer configuration.

In order to understand to code lines in this article it is necessary to view the entire process:

Pre running (the programmer)

1) Create an XML file that holds all the definitions of menu items and HTML tags / elements names.

2) Write client side scripts to handle the client side events for defining menu items. If the programmer sets the event of a menu item to be on the client side the mechanism will upload this event. The programmer should write the code to handle this event on the client side.

3) Write server side script to handle the server side events.

While running (the user)

4) The user right clicks on the page (it can be on any HTML tag on the page).

5) The Document\_on MouseDown event fires. The source element of the event, which actually fires the event and other data are sent to the web service via HTML components - HTC.

6) The web service checks the programmer XML file and constructs the HTML for the menu.

7) The client script that recalls the HTC receives the html and by using DHTML shows this to the user.

8) The user clicks on the menu.

9) Client side script checks the menu data and decides whether to upload client side script or server side script.

Data Design (XML Schema)

The Data Design should define the structure and data types for XML documents define on which side the event will be fired, and define arguments of the corresponding events. First, we define a simple type named ScriptLoc that keeps our event execute options (client or server). As explained above, the mechanism can upload the event when the user clicks on menu item on the server (server event) or on the client. Next, define the MenuItem type that consists of the name (the caption), argument (the programmer assigns argument that will be sent to the event), data (for future use) and ScriptLocation (one of the ScriptLoc values).

Now define the relation between HTML tags and element names of such tags. Build a general mechanism that will allow us to assign general-used menu items for every HTML tag and to assign specific menu items to specific HTML tag elements. These should have ID's so that they are unique. What we are achieving here is the containment relationship. To present it in a schema, we first define HtmlName type that consists of a Name (the id of the HTML item) and one or more MenuItems. Then we define the HtmlType type that is built from the Name (the TAG name), and none or collection of HtmlNames and one or more MenuItems. All the HtmlTypes are collected under a Page type that defines the WebForm name and collection of HtmlTypes.

The schema definition:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<xs:schema id="DynaMenuSchema" targetNamespace="DynaMenuSchema"
  elementFormDefault="qualified"
  xmlns:mstns="DynaMenuSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:simpleType name="ScriptLoc">
```

```

<xs:restriction base="xs:string">
  <xs:enumeration value="client" />
  <xs:enumeration value="server" />
</xs:restriction>
</xs:simpleType>

<xs:complexType name="MenuItem">
  <xs:sequence>

    <xs:element name="name" type="xs:string" />
    <xs:element name="argument" type="xs:string" />
    <xs:element name="data" type="xs:string" />
    <xs:element name="scriptloc" type="ScriptLoc" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="HtmlName">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="menuItem" type="MenuItem" minOccurs="1"
      maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="HtmlType">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="htmlname" type="HtmlName" minOccurs="0"

      maxOccurs="unbounded" />
    <xs:element name="menuItem" type="MenuItem" minOccurs="1"
      maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Page">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="htmltype" type="HtmlType" />
  </xs:sequence>
</xs:complexType>

<xs:element name="Pages">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="page" type="Page" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

```

## The Code

### Client side code

We begin with the simplest task -: to disable the default right-click behavior. To do so we need to catch the oncontextmenu to cancel its event bubble and to return false.

```

sub document_oncontextmenu()
  window.event.returnValue = false
  window.event.cancelBubble = true
end sub

```

We want to identify the right click event on any item on the screen. To do so, we take advantage of the event bubbling feature and catch the onmousedown event of the document. Using this technique we will be notified every time the user clicks on any of the page elements. This technique suits our requirement to enable the context menu for every HTML tag or HTML object with a name attribute.

At this stage we check if the user right-clicks and if a shown menu does or does not exist. By using the button property of the event we check for right or left click. Using the Contain method we check if the menu already exists.

```
if window.event.button = 2 and not window.document.body.contains
(document.body.children("DevNetmenu")) then
```

We are going to implement cache on the client side by using the dictionary. After catching the right click event and before calling the web service we use the ReturnCacheMenu function to check if we already have cached the menu data for the element with the given HTML tag or the given name attribute.

```
function ReturnCacheMenu(ItemTag,ItemName)
  RV = ""
  if (isempty(arrItemMenu)) then
    set arrItemMenu = createobject("Scripting.Dictionary")
    RV = ""
  else
    if (arrItemMenu.Exists(ItemName)) then
      RV = arrItemMenu.Item(ItemName)
    elseif (arrItemMenu.Exists(ItemTag)) then
      RV = arrItemMenu.Item(ItemTag)
    end if
  end
end
if
  ReturnCacheMenu = RV
end function
```

If the function finds cache data it is used to show the user the menu. Then we need to change the locations of the cached data so the menu will show the location where the user right clicked. If the key is not found in the dictionary we call the web service to receive the menu data. When the data arrives from the web service, we receive at the GetMenusResult parameter, the name of the tag or the name attribute that the web service used to create the menu. The return parameter will be used as the key that will be added to the dictionary together with the menu data.

If the menu data is not found in the cache we will call the web service. This process should begin by wiring the HTC. For this purpose we connect the HTC to the HTML element by using the BEHAVIOR in the style attribute.

```
<INPUT type="hidden" id="MenuHTC" style="BEHAVIOR:url(../webservice.htc);">
```

The next step is to use the useService function by supplying the web service URL and a friendly name. This friendly name is used to call the web service function with parameters.

We will call the web service asynchronously to prevent situations when there are large delays in producing the menu on the server side or when the server is unavailable. Further coding is required in order to supply the web service with the HTML tag and name attribute of some element. Some HTML elements are built from inner HTML tags (for example the TABLE element is built from TR and TD tags. If the user right-clicks on TD we ascend in the canonical hierarchy until we reach the TABLE level or the document level where we stop in order to prevent a situation called "orphan element" . This way we will send the TABLE tag to the web service and its name attribute right clicked by the user and this procedure saves the need to define menus to CAPTION, COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, and TR tags. In the code sample only the TABLE tag is handled, but the base table is enabled to hold an unlimited nested table.

The next step is to define a callback function in order to catch the answer from the web service, and to send the function address to the HTC. In the case of using VBScript, we employ the getref function to send the function address. The following is the coding implementation of the above.

```
sub document_onmousedown()
  if window.event.button = 2 and not window.document.body.contains
```

```

(document.body.children("DevNetmenu")) then
'need to check if its on the menu
dim Oelement

dim MenuData

set Oelement = window.event.srcElement

while (((Oelement.tagName <> "TABLE") or (Oelement.parentElement.tagName_
= "TD") ) and (Oelement.tagName <> "BODY"))

    set Oelement = Oelement.parentElement

wend
if Oelement.tagName <> "TABLE" then

    set Oelement = window.event.srcElement

end if

MenuData = ReturnCacheMenu(Oelement.tagName,Oelement.id)

if (MenuData = "") then
call window.document.all("MenuHTC").useService ( _
    "../DynaMenu.asmx?WSDL", "Menus")
iCallID = window.document.all("MenuHTC").Menus.callService(getref( _
    "menuhandle"), "GetMenus", Oelement.id , Oelement.tagName , _
    window.event.clientX , window.event.clientY , window.document.url, "")

else
'replace the previous location data with current location data

iLoc = instr(1,MenuData,"LEFT:")

MenuData = mid(MenuData,1,iLoc + len("LEFT:")) + " " + _
    cstr(window.event.clientX) + mid(MenuData,instr(iLoc+1,MenuData,"px;"))
iLoc = instr(1,MenuData,"TOP:")
MenuData = mid(MenuData,1,iLoc + len("TOP:")) + "" + _
    cstr(window.event.clientY) + mid(MenuData,instr(iLoc+1,MenuData,"px;"))
call document.body.insertAdjacentHTML ("beforeEnd",MenuData)

window.document.all("DevNetmenu").focus

end if
end if

end sub

```

#### Server side code

This section describes how to create a web service that handles the creation of the menu HTML corresponding to the programmer definitions in the XML file. Our first task is to create a Web service. Our action will be to add a web service to the existing project (instead of creating another project). This action will also help us to prevent refreshing of the entire page content when you need to change individual parts of the page. One of the innovations of the .NET is the PostBack property, however, the PostBack property concept is to create all the page content and render it on the client side. In order to add a web service to a project we need to add a new web project item to the project and to select the Web Service. The name of our web service will be DynaMenu.asmx.

The Web service will be built from a public function and several private functions. The public function will get the name of

the element as the by value first parameter, the second by value parameter will get the html type of the element, the third by value parameter will get the URL of the page and the fourth and the fifth will get the location where the user right-clicked and the last, will be a by-reference (ByRef) parameter which is required to modify the variable underlying the tag name or the name attribute that used to create the menu. All the other by value (ByVal) parameters will get their values from the XML file of the pre-defined menu items and will use these values to build context menu. The right-click location parameters(X,Y) will help us generate the HTML of the menu in the exact place where the user pointed (e right-clicked). In the attached code the Web service is implemented as part of the web application, but it is also available for creating the web service as a stand alone web service project. The code obtains the page name as one of its parameters so the form name with the path can be sent and the page name in the XML file with path also set .

The next step is to build the public function of the Web service. This function will be used to generate the DIV tag that will be the container for the displayed menu and set its property and events, in order that it will suit our needs (the context menu HTML). Following from that, the next step is to call the private function ProcessPage which is used to build the menu items. Next we will call the ProcessFixMenu function which is used to enable the addition of the menu default items,( those which will be permanent.) (Print, Move next, etc.)

```
<WebMethod()>Public Function GetMenus(ByVal ElemName As String,
ByVal ElemType As String, ByVal x As String, ByVal y As String,
ByVal page As String, ByRef ElementMenu As String) As String
    Dim oXmlReader As XPathDocument = Nothing
    Dim menu As System.Text.StringBuilder = New System.Text.StringBuilder()

    Dim PageName As String
```

To avoid reading the file of the disk every time we send an event we cache the XML file the first time we open it. Later on we will be able to work with cached objects.

```
If (Me.Context.Cache("MenuXml") Is Nothing) Then
    oXmlReader = New XPathDocument _
        (Me.Context.Request.PhysicalApplicationPath _
        + "\html\pages.xml")
    Me.Context.Cache.Insert("MenuXml",oXmlReader)
Else

    oXmlReader = Me.Context.Cache("MenuXml")
End If
```

The private method PageName is used literally to get the page name from the URL. This was added in order to remove the query string from the row URL. If you require to make this Web service stand alone, this should be changed to a function which returns a value of the page and its path.

```
PageName = GetPageName (page)
```

As mentioned previously, the menu that the web server produce is built from the DIV (context menu HTML) that serves as a block for the entire menu items. The DIV containing a table that holds the menu items as table cells (TD). First we will produce the DIV and its table, then each and every element in the XML file. , Matching the HTML tag and the name attribute of the given element we will produce a table cell that holds the caption and other data. While the DIV is produced we catch the OnBlur client side event and call to a client side script that handles the action followed by the user going out from the menu DIV. Next we set the location of the DIV to be at the location where the mouse down event occurred. Finally we will need to retain the element name on which the user right-clicked, in a hidden field, This data will be in a later use for firing the respective event at the client side.

```
menu.Append("<div onblur=""leaveMenu()"" onmouseout _
=""DevNetmenu.style.cursor = 'auto'"" onmouseover = _
""DevNetmenu.style.cursor = 'hand'"" id=""DevNetmenu"" _
name=""DevNetmenu"" style=""BORDER-TOP-STYLE: outset; _
BORDER-RIGHT-STYLE: outset; BORDER-LEFT-STYLE: outset;BORDER-BOTTOM-STYLE: _
outset;Z-INDEX:32000;BACKGROUND-COLOR: gray;LEFT: " + x + _
"px; WIDTH: 20px; POSITION: absolute; TOP: " _
+ y + "px; HEIGHT: 20px"">")
menu.Append ( "<INPUT name=""DevNetMenuItem"" ID=""DevNetMenuItem"" _
```

```
type=""hidden"" size=""6"" value="" + ElemName + ">")
menu.Append("<table id=DevNetmenuTbl name=DevNetmenuTbl>")
```

We continue by checking the existence of the page in the XML file, If the page exists we will take the definitions for the existing page. If not we will take definitions from a virtual page called &ldquo;general&rdquo;, a page that holds the default menu definitions. The processPage function will return the TD for the matched menu items and the HTML tag or attribute name that was used to produce the TD&rsquo;s.

```
If PageExist(PageName, oXmlReader) Then
'process the page
menu.Append(ProcessPage(ElemName, ElemType, PageName, oXmlReader, _
ElementMenu))

Else
menu.Append(ProcessPage(ElemName, ElemType, "general", oXmlReader, _
ElementMenu))

End If
```

Our last task will be to call the function responsible for adding the default menu items,, Because of their default nature they should be always displayed independently to the HTML tag or the attribute name.

```
'process fix menu
If menu.ToString().IndexOf("<tr>") <> -1 Then
menu.Append("<tr><td>-----</td></tr>")
End If
menu.Append(ProcessFixMenu())
```

```
menu.Append("</table></div>")

Return menu.ToString()
Catch err As
Exception

Context.Trace.Warn("Error", err.Message, err)
Finally
End Try
```

The PageExist function is a good example for the advantage of an XML path. A very simple yet elegant code can check if an element or attribute exists in the File. We will use an iteration to check all the elements in the file that match the Xpath query used before and with this iteration we can page through the matched elements.

```
Private Function PageExist(ByVal name As String, ByVal
oXmlReader As System.Xml.XPath.XPathDocument) As Boolean
Dim oPath As XPathNavigator
Dim olter As XPathNodeIterator
```

```
Dim bRetVal As Boolean = False
```

```
Try
```

```
oPath = oXmlReader.CreateNavigator()
```

The following code is very important. It handles the creation of the right queries so we can get the right answers or an simply an answer at all. For example, the query string, in the following code, searches for every page element in the XML with name attribute that has a value equal to the name parameter formerly received.

```
olter = oPath.Select("*/page[name="" + name + """]")
```

```
If olter.Count > 0 Then
```

```
bRetVal = True
```

```

End If
Catch Err As Exception

    Context.Trace.Warn("Error", err.Message, err)

Return False

End Try

End Function

```

Another role of the ProcessPage function is to use the XPath to search for a specified object name in the XML file. If the name exists its menu item is added as a TR to the table in the DIV block, with hidden data fields that we use in the client side script. After adding the menu items to the file we apply XPath query that searches for the specific HTML type. If a matched item is found its menu items are added to the table.

After using XPath to efficiently query the XML file with the result is an HTML string that represent a DIV with a table that holds the menu items for display together with hidden data that will be needed when we fire events (whenever the user selects one of the menu items.) This string is reverts back to the call back function that is needed to create in the client side script.

#### Handling the web service results in the client side

We begin by declaring a function that will be the call back function for the web service call. This was previously mentioned when we sent the address of this function. This function interface is fairly simple. It consists of one parameter which is the result of the calling web service function. In the body of the function we 1) check if the web-service has been completed successfully. 2) Use DHTML insertAdjacentHTML function to add the DIV HTML to the document so it will be visible to the end user. 3) Add the menu data by the ElementMenu a by reference (ByRef) parameter (hold the HTML type or name attribute that is used to create the menu) to the dictionary. 4) Set the focus to the menu DIV.

```

sub menuhandle(result)
    if (result.error) then

        msgbox "Error at the server side!"

    else
        call document.body.insertAdjacentHTML ("beforeEnd", _

            result.value.GetMenusResult)

        call arrItemMenu.add(result.value.ElementMenu, _
            result.value.GetMenusResult)

        window.document.all("DevNetmenu").focus

    end if
end sub

```

Now the menu is visible to the user who may select an item. The following code task is to catch this selection and to activate events on the client or the server side, depending on the programmer predefinition.

Before handling the user selections of menu item we need to handle the situation when the user clicks outside the DIV element. First of all when we add the menu DIV into the html we set the focus for him. That's mean that every click outside the DIV menu the OnBlur event will be raised. When we create the DIV html at the web service we assign the client side script leaveMenu to handle this event. implementaion is as follows: There is a simple trick to this function. We use the elementFromPoint to check where the user clicks. Now we have two possibilities. The first is that the user clicks on one of the menu items. In such a case the menu DIV loose its focus but we don't want to just remove the DIV, we want to handle the selection. The second possibility is that the user clicks outside the menu DIV. First we check if the clicked item is a table. If it's not a table we remove the DIV by using body.removeChild. If the clicked item is table we check if the table name matches the menu table. If there is a match we exit the function. If not, we remove the menu

```

sub leaveMenu()
  set oRes = window.document.elementFromPoint _
    (window.event.clientX,window.event.clientY)
  if oRes.Tagname = "TD" then
    if oRes.parentElement.parentElement.parentElement.ID = "DevNetmenuTbl" _
      then

      exit sub

    else

      set obj = document.body.children("DevNetmenu")

      document.body.removeChild(obj)

    end if

  else

    set obj = document.body.children("DevNetmenu")

    document.body.removeChild(obj)

  end if
end sub

```

To handle the selection of a menu we will implement the DevNetmenuTbl\_onclick that we assign to handle the function to the menu Table click event. When we create the menu html we assign a call to this function, with a parameter that resembles a menu command, to all the fixed menu elements and empty string parameters to all the dynamic menus. Now we will switch all the command parameters to the right functionality. Those menus are defined in advance and they resemble known functionality such as Move next, print or your own.

All the dynamic menu items (without command) are handled the same way. If the hidden field hidetextLoc, which exists under the TD element on which we click, and which holds the event location, indicates that the event should be on the server side. Then we fill the hidden fields holding the event data with the event name and with XML string. The XML data holds all the data that the programmer defines, and simulatesPostBack. The data is sent as an XML for two reasons: 1) to make the parsing process of data parts from the string easier. 2) If new data items need to be added to the string new code is not required in order to parse it. If the event should happen in the client we call a pre define function DevNetMenuHandle that the programmer is responsible to implement in order to process the event. This function has three parameters: the event argument, the event data and the element name that raises the event.

```

sub DevNetmenuTbl_onclick(sCommand)

  select case sCommand

    case "forward"

      call window.history.forward()

    case "back"

      call window.history.back()

    case "print"

      window.print ()
    case else
      if window.event.srcElement.all("hidetextLoc").value = "client" then

```

```

dim EventArg,EventValue,srcName

EventArg = window.event.srcElement.all("hidetextArg").value

EventValue = window.event.srcElement.all("hidetextData").value

srcName = window.document.all ("DevNetmenu").all _
("DevNetMenuItem").value

set obj = document.body.children("DevNetmenu")

document.body.removeChild(obj)
call DevNetMenuHandle(EventArg,EventValue,srcName)

else

```

We insert parameters to the hidden fields that are used byPostBack, so we can use them at the server side and to call up the desired event with the requested data. Microsoft uses this parameters to know who is the control that called up the event and which event should be raised. The `__EVENTTARGET` needs to be filled with the name of the web control that will raise the event. The `__EVENTARGUMENT` holds the name of the event that the web control should fire. In most cases when the webcontrol has only one event there is no need for this parameter. with this knowledge you can simulate PostBack of web controls from script on the client side. In the code a new form is created and the data of the PostBack hidden fields changed, because of VBScript limitation. Microsoft renders to the client JavaScript function `__doPostBack()` and the attaches it to the HTML tags event with the right parameters. The problem with VBScript is that it cannot call functions that start with underscore. One workaround to this problem is replacing the javascript function.

```

set theform = document.frmMain
theform.all("__EVENTTARGET").value = "devnetmenu"
theform.all("__EVENTARGUMENT").value = "<?xmlversion=""1.0"" _
encoding=""utf-8"" ?><EventData><arg>" + _
window.event.srcElement.all("hidetextArg").value + _
"</arg><name>" + window.document.all ("DevNetmenu").all _
("DevNetMenuItem").value + "</name><data>" + _
window.event.srcElement.all("hidetextData").value + _
"</data></EventData>"

call theform.submit()

end if

end select

set obj = document.body.children("DevNetmenu")

document.body.removeChild(obj)

end sub

sub DevNetMenuHandle(EventArg,EventValue,srcName)
msgbox "r.click on " + srcName + " with arg = " + EventArg + _
" and data = " + EventData

end sub

```

In this paragraph we handled the user selection of menu item or the clicking outside of the menu. If the event was set to be in the client we also showed how to implement it. In the next section we will show who to handle the event in the server and how to implement most of the client script on the server side.

Handle the event on the server side and encapsulate the client script

The server side code is actually designed for every web form that we are going to write. So there is good reason to overload the default Page class with our class which will implement all our specific tasks. This way every page that will inherit from our page class (that inherits from System.Web.UI.Page) can enjoy all the benefits that we put into it.

The first step will be to add class library project to our solution. We will implement our page class in a different assembly so that every page can reference it and inherit its page classes from it. We will name the project as DevOurPage and the class as DevPage. we begin by developing this assembly with C# for two reasons: 1) to show the simplicity of combining different language in the same solution. 2) the C# language is more suited for writing infrastructure, because of the need to use unmanaged code.

After adding a class library project, named DevOurPage to our solution we name the Class as DevPage, and add a reference to System.Web.UI. using the Using key word, this will give our class all the functionality that the Web.UI.Page class possesses as we will need to inherit from it. Don't forget to call up the base class constructor.

We now need a mechanism to catch the requests that are coming from the client side (viaPostBack) for Menu events and raise the menu event on the server side. To do this we need to do two things. firstly, we declare an event that we can raise. Secondly, we need to catch the client request for server event and raise it.

Declaring the menu event

To declare new event arguments we need to create new class that inherits from System.EventArgs, (the default class for event parameters). In this, way we can transfer our parameters to the event. We need to transfer the event name, its argument and its data. So we 1) declare three private members. 2) Create a new constructor that receives those members as parameters and sets the inner private data. 3) Create public property that will set and get parameters.

After creating class that will hold our arguments we need to declare delegation on our page, that will accept as a parameter only instances of the SelChangeMenu class. Following this, we can declare the event as public member of our base page.

[Serializable]

```
public class SelChangeMenu : EventArgs
```

```
{
    string _Args;
    string _Name;
    string _Data;

    public SelChangeMenu()
    {
    }

    public SelChangeMenu(string args,string name,string data)
    {
        _Args = args;
        _Name = name;
        _Data = data;
    }

    public string Args
    {
```

```
get
{
    return _Args;
}
set
{
    _Args = value;
}

}

public string Name
{
    get
    {
        return _Name;
    }
    set
    {
        _Name = value;
    }
}

public string Data
{
    get
    {
        return _Data;
    }
    set
    {
        _Data = value;
    }
}
}
```

```
}
```

```
public class DevPage : Page
```

```
{
[Serializable]
public delegate void MenuEventHandler(object sender, SelChangeMenu e);
public event MenuEventHandler MenuClick;
```

Catch the client request and raise the event

The data sent from the client is received at the server side in the form collection of the Request object. In the form collection there are fields that start with an underline. Those fields are basically internal and hold the data that the server sends to the client and vice versa. These fields include the `__<CODE>EVENTTARGET` and `__<CODE>EVENTARGUMENT` field. The first holds the name of the control that should be raised and the second holds the event. MS catches the request for the event and raises the appropriate event for us. When we create an UI control we implement the `RaisePostBackEvent` function of the `IPostBackEventHandler` interface to handle the event. But in this case we create a new event. The page default event handler is unsure of what to do with event that was post back with `__<CODE>EVENTTARGET` and set to `devnetmenu`. This must therefore be handled. simply by overriding the `OnInit` member of the base page class.

In the `OnInit` function we can check if the `__EVENTTARGET` set to `devnetmenu`. If so we can parse the XML that was sent from the client in the `__EVENTARGUMENT` and raise the server event with the parameters that we declared earlier.

```
override protected void OnInit(EventArgs e)
```

```
{
if (this.Request.Form["__EVENTTARGET"] == "devnetmenu")

{
System.IO.StringReader oReader = new _
System.IO.StringReader(this.Request.Form["__EVENTARGUMENT"]);

System.Xml.XPath.XPathDocument oDoc = new _
System.Xml.XPath.XPathDocument(oReader);

System.Xml.XPath.XPathNavigator oPath = oDoc.CreateNavigator();

System.Xml.XPath.XPathNodeIterator olter;

olter = oPath.Select("*/arg");

olter.MoveNext() ;

string arg = olter.Current.Value;

olter = oPath.Select("*/name");

olter.MoveNext() ;

string name = olter.Current.Value;

olter = oPath.Select("*/data");

olter.MoveNext() ;

string data = olter.Current.Value;
```

```
SelChangeMenu oEvent = new SelChangeMenu(arg,name,data);
```

We must check if the event handler is null to prevent InvalidUseOfNull exception. We will come across this situation if the programmer failed to create a menu handler on the page.

```
if (MenuClick != null)
```

```
    MenuClick(this,oEvent);
```

```
}
```

In order to prevent an initiation of the default behavior of the function, we now call the base class.

```
    base.OnInit (e);
```

```
}
```

Now we implement the event so we can catch the events and write our own code to handle them. To do this we implement the event on the page on which we are working, (which is actually the same page that will inherit from our page class.)

```
Private Sub Page_MenuClick(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles MyBase.MenuClick  
    Response.Write("menu click at " + CType(e, DevOurPage.SelChangeMenu).Name)
```

```
End Sub
```