

Re: All Menu Navigation

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.aspnet/2007-01/msg02764.html>

- *From:* "Tim Mackey" <tim.mackey@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 23 Jan 2007 11:45:56 -0000
-

hi john,

i'm also interested in designing proper navigation for web sites with clean markup. i don't like the built-in .net menu control because it generates bloated markup.

although i haven't heard of the 'all-menu' before, i have implemented one just like it recently. according to the all-menu description, it is fully expanded all the time, i.e. static. in this case you would have no need for ajax or javascript or dropdowns. my implementation below is static, pure html, styled with css.

if you want to look at a sample, i have uploaded a test page to: http://tim.mackey.ie/stuff/Sample_Menu.html.

if you view the source you'll see that it generates nice markup, feel free to use the styles etc as you see fit.

i based the implementation on sample code from <http://www.asp.net/CSSAdapters/Menu.aspx>, which shows how to create decent HTML for a Menu control bound to a SiteMapDataSource. the example above is dynamic/dropdown-based, but I wanted a static menu so i modified it. i would recommend you follow the example mentioned above and get it integrated into your site and working. i've included my modified version if you are interested, you can just replace MenuAdapter.cs with the code below. although it looks like a lot of code just to render a menu, it isn't really. the internal code for the built-in asp.net menu control would look very similar. it runs lightning fast for me anyway.

```
using System;
using System.Data;
using System.Collections;
using System.Collections.Generic;
using System.Configuration;
using System.Reflection;
using System.Web;
using System.Web.Configuration;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.Adapters;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.Adapters;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
```

```
public class WebControlAdapterExtender
{
    private WebControl _adaptedControl = null;
    public WebControl AdaptedControl
    {
```

Re: All Menu Navigation

```
get
{
System.Diagnostics.Debug.Assert(_adaptedControl != null, "CSS Friendly adapters internal error", "No
control has been defined for the adapter extender");
return _adaptedControl;
}
}

public bool AdapterEnabled
{
get
{
bool bReturn = true; // normally the adapters are enabled

// Individual controls can use the expando property called AdapterEnabled
// as a way to turn the adapters off.
// <asp:TreeView runat="server" AdapterEnabled="false" />
if ((AdaptedControl != null) &&
(!String.IsNullOrEmpty(AdaptedControl.Attributes["AdapterEnabled"])) &&
(AdaptedControl.Attributes["AdapterEnabled"].IndexOf("false", StringComparison.OrdinalIgnoreCase) ==
0))
{
bReturn = false;
}

return bReturn;
}
}

private bool _disableAutoAccessKey = false; // used when dealing with things like read-only textboxes that
should not have access keys
public bool AutoAccessKey
{
get
{
// Individual controls can use the expando property called AdapterEnabled
// as a way to turn on/off the heuristic for automatically setting the AccessKey
// attribute in the rendered HTML. The default is shown below in the initialization
// of the bReturn variable.
// <asp:TreeView runat="server" AutoAccessKey="false" />

bool bReturn = true; // by default, the adapter will make access keys are available
if (_disableAutoAccessKey ||
((AdaptedControl != null) &&
(!String.IsNullOrEmpty(AdaptedControl.Attributes["AutoAccessKey"])) &&
(AdaptedControl.Attributes["AutoAccessKey"].IndexOf("false", StringComparison.OrdinalIgnoreCase) ==
0)))
{
bReturn = false;
}
}
```

Re: All Menu Navigation

```
return bReturn;
}
}
```

```
public WebControlAdapterExtender(WebControl adaptedControl)
{
    _adaptedControl = adaptedControl;
}
```

```
public void RaiseAdaptedEvent(string eventName, EventArgs e)
{
    string attr = "OnAdapted" + eventName;
    if ((AdaptedControl != null) && (!String.IsNullOrEmpty(AdaptedControl.Attributes[attr])))
    {
        string delegateName = AdaptedControl.Attributes[attr];
        Control methodOwner = AdaptedControl.Parent;
        MethodInfo method = methodOwner.GetType().GetMethod(delegateName);
        if (method == null)
        {
            methodOwner = AdaptedControl.Page;
            method = methodOwner.GetType().GetMethod(delegateName);
        }
        if (method != null)
        {
            object[] args = new object[2];
            args[0] = AdaptedControl;
            args[1] = e;
            method.Invoke(methodOwner, args);
        }
    }
}
```

```
public void RenderBeginTag(HtmlTextWriter writer, string cssClass)
{
    string id = (AdaptedControl != null) ? AdaptedControl.ClientID : "";

    if (!String.IsNullOrEmpty(AdaptedControl.Attributes["CssSelectorClass"]))
    {
        WriteBeginDiv(writer, AdaptedControl.Attributes["CssSelectorClass"], id);
        id = "";
    }

    WriteBeginDiv(writer, cssClass, id);
}
```

```
public void RenderEndTag(HtmlTextWriter writer)
{
    WriteEndDiv(writer);
}
```

Re: All Menu Navigation

```
if (!String.IsNullOrEmpty(AdaptedControl.Attributes["CssSelectorClass"]))
{
    WriteEndDiv(writer);
}
}
```

```
static public void RemoveProblemChildren(Control ctrl, List<ControlRestorationInfo> stashedControls)
{
    RemoveProblemTypes(ctrl.Controls, stashedControls);
}
```

```
static public void RemoveProblemTypes(ControlCollection coll, List<ControlRestorationInfo>
stashedControls)
{
    foreach (Control ctrl in coll)
    {
        if (typeof(RequiredFieldValidator).IsAssignableFrom(ctrl.GetType()) ||
            typeof(CompareValidator).IsAssignableFrom(ctrl.GetType()) ||
            typeof(RegularExpressionValidator).IsAssignableFrom(ctrl.GetType()) ||
            typeof(ValidationSummary).IsAssignableFrom(ctrl.GetType()))
        {
            ControlRestorationInfo cri = new ControlRestorationInfo(ctrl, coll);
            stashedControls.Add(cri);
            coll.Remove(ctrl);
            continue;
        }
    }
}
```

```
if (ctrl.HasControls())
{
    RemoveProblemTypes(ctrl.Controls, stashedControls);
}
}
}
```

```
static public void RestoreProblemChildren(List<ControlRestorationInfo> stashedControls)
{
    foreach (ControlRestorationInfo cri in stashedControls)
    {
        cri.Restore();
    }
}
```

```
public string MakeChildId(string postfix)
{
    return AdaptedControl.ClientID + "_" + postfix;
}
```

```
static public string MakeNameFromId(string id)
{
    string name = "";
    for (int i=0; i<id.Length; i++)
```

Re: All Menu Navigation

```
{
char thisChar = id[i];
char prevChar = ((i - 1) > -1) ? id[i - 1] : '';
char nextChar = ((i + 1) < id.Length) ? id[i + 1] : '';
if (thisChar == '_')
{
if (prevChar == '_')
{
name += "_";
}
else if (nextChar == '_')
{
name += "$_";
}
else
{
name += "$";
}
}
else
{
name += thisChar;
}
}
return name;
}

static public string MakeIdWithButtonType(string id, ButtonType type)
{
string idWithType = id;
switch (type)
{
case ButtonType.Button:
idWithType += "Button";
break;
case ButtonType.Image:
idWithType += "ImageButton";
break;
case ButtonType.Link:
idWithType += "LinkButton";
break;
}
return idWithType;
}

public string MakeChildName(string postfix)
{
return MakeNameFromId(MakeChildId(postfix));
}

static public void WriteBeginDiv(HtmlTextWriter writer, string className, string id)
```

Re: All Menu Navigation

```
{
writer.WriteLine();
writer.WriteBeginTag("div");
if (!String.IsNullOrEmpty(className))
{
writer.WriteAttribute("class", className);
}
if (!String.IsNullOrEmpty(id))
{
writer.WriteAttribute("id", id);
}
writer.Write(HtmlTextWriter.TagRightChar);
writer.Indent++;
}
```

```
static public void WriteEndDiv(HtmlTextWriter writer)
{
writer.Indent--;
writer.WriteLine();
writer.WriteEndTag("div");
}
```

```
static public void WriteSpan(HtmlTextWriter writer, string className, string content)
{
if (!String.IsNullOrEmpty(content))
{
writer.WriteLine();
writer.WriteBeginTag("span");
if (!String.IsNullOrEmpty(className))
{
writer.WriteAttribute("class", className);
}
writer.Write(HtmlTextWriter.TagRightChar);
writer.Write(content);
writer.WriteEndTag("span");
}
}
```

```
static public void WriteImage(HtmlTextWriter writer, string url, string alt)
{
if (!String.IsNullOrEmpty(url))
{
writer.WriteLine();
writer.WriteBeginTag("img");
writer.WriteAttribute("src", url);
writer.WriteAttribute("alt", alt);
writer.Write(HtmlTextWriter.SelfClosingTagEnd);
}
}
```

```
static public void WriteLink(HtmlTextWriter writer, string className, string url, string title, string content)
```

Re: All Menu Navigation

```
{
if ((!String.IsNullOrEmpty(url)) && (!String.IsNullOrEmpty(content)))
{
writer.WriteLine();
writer.WriteBeginTag("a");
if (!String.IsNullOrEmpty(className))
{
writer.WriteAttribute("class", className);
}
writer.WriteAttribute("href", url);
writer.WriteAttribute("title", title);
writer.Write(HtmlTextWriter.TagRightChar);
writer.Write(content);
writer.WriteEndTag("a");
}
}

// Can't be static because it uses MakeChildId
public void WriteLabel(HtmlTextWriter writer, string className, string text, string forId)
{
if (!String.IsNullOrEmpty(text))
{
writer.WriteLine();
writer.WriteBeginTag("label");
writer.WriteAttribute("for", MakeChildId(forId));
if (!String.IsNullOrEmpty(className))
{
writer.WriteAttribute("class", className);
}
writer.Write(HtmlTextWriter.TagRightChar);

if (AutoAccessKey)
{
writer.WriteBeginTag("em");
writer.Write(HtmlTextWriter.TagRightChar);
writer.Write(text[0].ToString());
writer.WriteEndTag("em");
if (!String.IsNullOrEmpty(text))
{
writer.Write(text.Substring(1));
}
}
else
{
writer.Write(text);
}

writer.WriteEndTag("label");
}
}
```

Re: All Menu Navigation

```
// Can't be static because it uses MakeChildId
public void WriteTextBox(HtmlTextWriter writer, bool isPassword, string labelClassName, string labelText,
string inputClassName, string id, string value)
{
    WriteLabel(writer, labelClassName, labelText, id);

    writer.WriteLine();
    writer.WriteBeginTag("input");
    writer.WriteAttribute("type", isPassword ? "password" : "text");
    if (!String.IsNullOrEmpty(inputClassName))
    {
        writer.WriteAttribute("class", inputClassName);
    }
    writer.WriteAttribute("id", MakeChildId(id));
    writer.WriteAttribute("name", MakeChildName(id));
    writer.WriteAttribute("value", value);
    if (AutoAccessKey && (!String.IsNullOrEmpty(labelText)))
    {
        writer.WriteAttribute("accesskey", labelText[0].ToString().ToLower());
    }

    writer.Write(HtmlTextWriter.SelfClosingTagEnd);
}

// Can't be static because it uses MakeChildId
public void WriteReadOnlyTextBox(HtmlTextWriter writer, string labelClassName, string labelText, string
inputClassName, string value)
{
    bool oldDisableAutoAccessKey = _disableAutoAccessKey;
    _disableAutoAccessKey = true;

    WriteLabel(writer, labelClassName, labelText, "");

    writer.WriteLine();
    writer.WriteBeginTag("input");
    writer.WriteAttribute("readonly", "readonly");
    if (!String.IsNullOrEmpty(inputClassName))
    {
        writer.WriteAttribute("class", inputClassName);
    }
    writer.WriteAttribute("value", value);
    writer.Write(HtmlTextWriter.SelfClosingTagEnd);

    _disableAutoAccessKey = oldDisableAutoAccessKey;
}

// Can't be static because it uses MakeChildId
public void WriteCheckBox(HtmlTextWriter writer, string labelClassName, string labelText, string
inputClassName, string id, bool isChecked)
{
    writer.WriteLine();
```

Re: All Menu Navigation

```
writer.WriteBeginTag("input");
writer.WriteAttribute("type", "checkbox");
if (!String.IsNullOrEmpty(inputClassName))
{
writer.WriteAttribute("class", inputClassName);
}
writer.WriteAttribute("id", MakeChildId(id));
writer.WriteAttribute("name", MakeChildName(id));
if (isChecked)
{
writer.WriteAttribute("checked", "checked");
}
if (AutoAccessKey && (!String.IsNullOrEmpty(labelText)))
{
writer.WriteAttribute("accesskey", labelText[0].ToString());
}
writer.Write(HtmlTextWriter.SelfClosingTagEnd);

WriteLabel(writer, labelClassName, labelText, id);
}

// Can't be static because it uses MakeChildId
public void WriteSubmit(HtmlTextWriter writer, ButtonType buttonType, string className, string id, string
imageUrl, string javascript, string text)
{
writer.WriteLine();

string idWithType = id;

switch (buttonType)
{
case ButtonType.Button:
writer.WriteBeginTag("input");
writer.WriteAttribute("type", "submit");
writer.WriteAttribute("value", text);
idWithType += "Button";
break;
case ButtonType.Image:
writer.WriteBeginTag("input");
writer.WriteAttribute("type", "image");
writer.WriteAttribute("src", imageUrl);
idWithType += "ImageButton";
break;
case ButtonType.Link:
writer.WriteBeginTag("a");
idWithType += "LinkButton";
break;
}

if (!String.IsNullOrEmpty(className))
{
```

Re: All Menu Navigation

```
writer.WriteAttribute("class", className);
}
writer.WriteAttribute("id", MakeChildId(idWithType));
writer.WriteAttribute("name", MakeChildName(idWithType));

if (!String.IsNullOrEmpty(javascript))
{
    string pureJS = javascript;
    if (pureJS.StartsWith("javascript:"))
    {
        pureJS = pureJS.Substring("javascript:".Length);
    }
    switch (buttonType)
    {
        case ButtonType.Button:
            writer.WriteAttribute("onclick", pureJS);
            break;
        case ButtonType.Image:
            writer.WriteAttribute("onclick", pureJS);
            break;
        case ButtonType.Link:
            writer.WriteAttribute("href", javascript);
            break;
    }
}

if (buttonType == ButtonType.Link)
{
    writer.Write(HtmlTextWriter.TagRightChar);
    writer.Write(text);
    writer.WriteEndTag("a");
}
else
{
    writer.Write(HtmlTextWriter.SelfClosingTagEnd);
}
}

static public void WriteRequiredFieldValidator(HtmlTextWriter writer, RequiredFieldValidator rfv, string
className, string controlToValidate, string msg)
{
    if (rfv != null)
    {
        rfv.CssClass = className;
        rfv.ControlToValidate = controlToValidate;
        rfv.ErrorMessage = msg;
        rfv.RenderControl(writer);
    }
}

static public void WriteRegularExpressionValidator(HtmlTextWriter writer, RegularExpressionValidator rev,
```

Re: All Menu Navigation

```
string className, string controlToValidate, string msg, string expression)
{
if (rev != null)
{
rev.CssClass = className;
rev.ControlToValidate = controlToValidate;
rev.ErrorMessage = msg;
rev.ValidationExpression = expression;
rev.RenderControl(writer);
}
}
```

```
static public void WriteCompareValidator(HtmlTextWriter writer, CompareValidator cv, string className,
string controlToValidate, string msg, string controlToCompare)
{
if (cv != null)
{
cv.CssClass = className;
cv.ControlToValidate = controlToValidate;
cv.ErrorMessage = msg;
cv.ControlToCompare = controlToCompare;
cv.RenderControl(writer);
}
}
```

```
static public void WriteTargetAttribute(HtmlTextWriter writer, string targetValue)
{
if ((writer != null) && (!String.IsNullOrEmpty(targetValue)))
{
// If the targetValue is _blank then we have an opportunity to use attributes other than "target"
// which allows us to be compliant at the XHTML 1.1 Strict level. Specifically, we can use a combination
// of "onclick" and "onkeypress" to achieve what we want to achieve when we used to render
// target='blank'.
//
// If the targetValue is other than _blank then we fall back to using the "target" attribute.
// This is a heuristic that can be refined over time.
if (targetValue.Equals("_blank", StringComparison.OrdinalIgnoreCase))
{
string js = "window.open(this.href, '_blank', ''); return false;";
writer.WriteAttribute("onclick", js);
writer.WriteAttribute("onkeypress", js);
}
else
{
writer.WriteAttribute("target", targetValue);
}
}
}
}
```

```
public class ControlRestorationInfo
```

Re: All Menu Navigation

```
{
private Control _ctrl = null;
public Control Control
{
get { return _ctrl; }
}

private ControlCollection _coll = null;
public ControlCollection Collection
{
get { return _coll; }
}

public bool IsValid
{
get { return (Control != null) && (Collection != null); }
}

public ControlRestorationInfo(Control ctrl, ControlCollection coll)
{
_ctrl = ctrl;
_coll = coll;
}

public void Restore()
{
if (IsValid)
{
_coll.Add(_ctrl);
}
}
}
```

hope you like it
tim

"John" <JohnSickOfSpam@xxxxxxx> wrote in message
news:em50yelPHHA.856@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Hi there,

I was reading an article
(http://avenuea-razorfish.com/articles/TheAll-MenuNavigation_Turbek.pdf) on 'all-menu navigation' and I'd like to try and implement this in my site. Can anyone recommend how to get started?

Is there an Ajax control available for this or would more a basic javascript work just as well. I'm assuming I just need a div element that is shown on a mouse over, but I'm not sure how to go about aligning all of the sub menu items etc?

Re: All Menu Navigation

Please help a newbie!

Best regards

John