Dieser Artikel ist in der Ausgabe 05/2008 von ASP.NET professional zu finden, diese Veröffentlichung geschieht mit freundlich Genehmigung von [ASP.NET professional]().

StyleSwitcher selbst gemacht

Stylesheets einer ASP.NET Anwendung on-the-fly Austauschen

Peter Bucher

Eine Webanwendung kommt heutzutage nicht mehr ohne Cascading Stylesheets (CSS) aus.
Durch diese Technologie, die Formatierung und Daten trennt, ergibt sich für Sie auch die Möglichkeit, ganze Layouts einer Seite auszutauschen, ohne den (X)HTML-Code selbst anzufassen. Es gibt viele Websites, die schon seit längerem eine Umschaltung von Layouts unterstützen.
Diese Umschaltung kann mithilfe von CSS -Dateien oder aber auch mithilfe von Bildern und Inline-Style (Eingebettete Stylesheet-Deklarationen) realisiert sein.

Eine Lösung mit externen CSS-Dateien ist auf alle Fälle zu bevorzugen, weil Sie sich so an die Regel und den Sinn von CSS halten: Layout und Daten zu trennen.

Im Laufe dieses Artikels erfahren Sie die Gedanken hinter einem solchen Control,
die Funktionsweise wird erläutert und die Implementation Schritt für Schritt durchgegangen.
Dabei nimmt die vorgestellte Lösung auf wichtige Aspekte Rücksicht, die vielfach nicht beachtet werden.
Ein Beispiel mit diesem Control in Aktion können Sie unter [2] ansehen.

Diese wären unter anderem:

- Komplette Funktionalität ist client- sowie auch serverseitig vorhanden
- Realisierung mit Rücksicht auf Browser, die kein Javascript verstehen. Zusätzlich optionale Javascript-Unterstützung als Komfortfunktion.
- Semantisch korrektes Html bei der Ausgabe des User Interfaces, dadurch eine hohe Flexibilität für das Aussehen per CSS-Formatierungen.
- Mehrere Instanzen dieser Lösung auf einer Domain kommen sich nicht in die Quere

Die gesamte Implementierung erfolgt als ASP.NET Server-Control. Somit stellt sie eine einfach anzuwendende und wiederverwendbare Komponente dar.

Grundüberlegungen

Die Lösung wird darauf ausgelegt, dass ein Layout jeweils von einer CSS-Datei repräsentiert wird. Gibt es Layout-Informationen die bei jedem Layout gleichbleibend sind, können diese manuell durch ein zusätzliches <link>-Tag in der Seite eingebunden werden.

In einem Ordner der Webanwendung liegen alle verfügbaren Layouts, die dann ausgewählt werden können. Diese verfügbaren Layouts werden beim Laden des Controls automatisch anhand eines angegebenen Basispfades ausgelesen.
Optional können die Layouts auch per Code, deklarativ über ASP.NET-Code oder per Datenbindung an das Control gebunden werden.

Die Client- und Serverseite Um eine Umschaltung per Javascript zu ermöglichen, gibt es ein clientseitiges Script, dass das „href“-Attribut des Link-Tags per DOM (Document Object Model) austauscht. Dies veranlasst den Browser im Hintergrund, einen GET-Request (Anforderung an den Server per HTTP Verb „GET“) zu tätigen und die benötigten Daten zu laden. Nach dem Laden wird das Layout sofort auf die aktuelle Seite angewandt.

Um das clientseitig ausgewählte Layout auch bei einer erneuten Seitenanforderung an den Server beizubehalten, wird per Javascript ein Cookie gesetzt, dass dann auf der Serverseite ausgewertet und gegebenfalls angewendet wird.

Genau so wird serverseitig bei einer Auswahl das gleiche Cookie geschrieben und gegebenfalls überschrieben. Das gesetzte Cookie hält die Daten auf Client- und Serverseite, beziehungsweise Javascript und ASP.NET aktuell.

Wenn clientseitig per Javascript das Layout geändert worden ist und anschließend eine andere Seite angefordert wird, setzt das Control serverseitig automatisch den aktuellen Wert im Cookie.
Das heißt es gibt keine unnötige Javascript-Aktivität bei einem erneuten Laden der Seite.

Um auch ohne Javascript die volle Funktionalität zu gewährleisten, wird ein Link in der Layout-Aufzählungsliste aufgebaut (siehe Listing 1).

Listing 1: Beispiel-Link

```
<li>
    <a href="?style=someLayout.css" onclick="return switchStyle(this,
'someLayout.css');">someLayout</a>
</li>
```

Mit Javascriptunterstützung

Dadurch, dass zusätzlich zum „onclick“-Attribut auch das „href“-Attribut genutzt wird, stehen beide Möglichkeiten (client- oder serverseitige Aktion) offen.
Wenn Javascript vorhanden ist, wird die „switchStyle()“-Methode aufgerufen, die clientseitige Aktion erledigt und anschliessend „false“ zurückgegeben (siehe Listing 9).

Ist das der Fall und „false“ wird zurückgegeben, hat sich die Sache für den Browser erledigt.
Durch „return false;“ wird der Link hinter dem „href“-Attribut nicht mehr aufgerufen.

Ohne Javascriptunterstützung

Wird ein derart aufgebauter Link in einem Browser ohne Javascriptunterstützung angeklickt, kommt der Javascript-Teil natürlich nicht zum Zug. Stattdessen wird der Link aufgerufen und infolgedessen ein GET-Request an den Server gesendet, der im QueryString die Information für das gewollte Layout mitbringt. Dabei ist noch zu beachten, dass möglicherweise schon vorhandene QueryString-Parameter mit in den Link aufgenommen werden. So ist gewährleistet, dass die Seite, in der das Control eingebunden wird, auch ohne Probleme arbeiten kann. Diese Aufgabe wird durch eine Hilfsmethode erledigt, die auch Bestandteil des kompletten Quellcodes ist.

Weitere Probleme – und wie Sie diese loswerden

Javascript und ASP.NET benutzen das gleiche Cookie, indem sie beide mit dem gleichen Namen darauf zugreifen. Bei einem normalen Einsatz wird das Cookie beispielsweise „styleSwitcher“ heißen. Dies führt aber zu Problemen, wenn mehrere Applikationen auf einer Domain laufen. Dann wird das Cookie unter den Anwendungen geteilt, das heißt nach einem Umschalten in der einen Applikation hat die zweite Applikation den gleichen Wert im Cookie.

Um das zu verhindern, wird ein eindeutiger Name für das Cookie benötigt. Die einfachste Art, dies zu erreichen, ist die Übergabe von einem Anwendungsnamen an das Control. Das Control setzt dann intern den Cookienamen wie folgt zusammen: *„styleSwitcher_“ + <Applikationsname>*

Wahl der Basisklasse für das Control

Bei der Wahl der Basisklasse sollte diese möglichst nah an das Ziel kommen, aber im Optimalfall auch nicht mehr bieten, als für das Ziel notwendig ist. Vom Control soll primär eine Liste von Links ausgegeben werden, die dann beim Anklicken zu der entsprechenden Layout-Änderung führen.
Als Linkliste eignet sich – im Zeitalter von CSS – eine ungeordnete Liste (ul- und li-Tags) und darin verschachtelte Links (Anker-Tags).

Als optionalen Zusatz wird eine Enumeration eingeführt, um zwischen drei verschiedenen Ausgabemodi zu wählen.

- Ungeordnete Liste mit Links (List)
- DropDown-Auswahlfeld (DropDown)
- Kombination der beiden Möglichkeiten (Combined)

Diese Anforderung lässt sich ganz elegant mit der ASP.NET- Klasse „ListItem“ lösen, die von mehreren ähnlichen Controls benutzt wird. Als Basisklasse kommt die abstrakte Basisklasse „ListControl“ zum Einsatz. Diese stellt die Grundimplementation eines datengebundenen List-Controls bereit, also genau das, was hier benötigt wird. Sie rendert per Standard eine DropDown-Liste. Dieses Verhalten wird genutzt, um den zweiten Ausgabemodus (oben) zu unterstützen. Die Ausgabe des ersten Ausgabemodus wird anschließend per Hand in der überschriebenen „Render“-Methode des Controls erledigt. Dabei kann auf die gleiche ListItem-Collection zugegriffen werden, was die Sache ganz einfach macht.

Die Implementierung im Detail betrachtet

Listing 2: Benötigte Instanzvariablen

```
private string        _applicationName;
private string        _currentStyle;
```

```
private string        _basePath;
private string        _cssPath;
private string        _activeCssClass;
private bool          _styleShowEnabled;
private RenderingMode _renderingMode;
```

Alle benötigten Instanzvariablen werden über Eigenschaften nach außen bereitgestellt. Folgend eine kurze Übersicht über den Zweck der Variablen (siehe Listing 2):

ApplicationName

- Muss gesetzt werden, um Kollisionen von Cookies auf der gleichen Domain zu verhindern.
- Wird intern benutzt, um die Cookienamen einzigartig zu vergeben.

CurrentStyle

- Über diese Eigenschaft kann das aktuell gesetzte Layout abgerufen werden.
- Wird hauptsächlich intern genutzt.

BasePath

- Der Basispfad bestimmt den Basisstandort der CSS-Dateien, die benutzt werden. Falls die Dateien direkt im Root liegen, kann dieser weggelassen werden.

CssPath

- Wenn diese Eigenschaft gesetzt wird, lädt das Control automatisch alle Layouts aus dem Ordner aus und benutzt diese.

ActiveCssClass

- Legt die zu benutzende CSS-Klasse fest, um die aktive Layout-Wahl hervorzuheben.

StyleShowEnabled

- Standardmäßig auf „false“, legt fest, ob die Option angeboten werden soll, Layouts schrittweise nachzuladen. Dazu später mehr.

RenderingMode

- Gibt die Möglichkeit, zwischen den drei oben genannten Ausgabemodi zu wählen.

Der Standardkonstruktor ohne Parameter wird immer aufgerufen und setzt Standardwerte fest. Über den zweiten Konstruktor kann der Basispfad festgelegt werden und im dritten Konstruktor optional noch das Standard-Layout (siehe Listing 3).

Listing 3: Konstruktoren

```
public StyleSwitcher() {
    this._renderingMode = RenderingMode.List;
    this._styleShowEnabled = false;
}

public StyleSwitcher(string basePath) : this() {
```

```
    this._basePath = basePath;
}

public StyleSwitcher(string basePath, string currentStyle) : this(basePath)
{
    this._currentStyle = currentStyle;
}
```

Es wird zusätzlich eine Enumeration bereitgestellt, diese wird mit der Eigenschaft „RenderingMode“ benutzt, um zwischen den drei verfügbaren Modi zu unterscheiden (Siehe Listing 4).

Listing 4: Enumeration für die drei Ausgabemodi

```
public enum RenderingMode {
    List,
    DropDown,
    Combined
};
```

In Kürze erledigt die überschriebene OnLoad Methode folgende Schritte: Ein eventuell vorhandenen Cookie wird ausgelesen und bei Vorhandensein der Wert davon als aktuelles Layout gesetzt. Anschließend wird der QueryString ausgelesen. Falls vorhanden den Wert davon als aktuelles Layout setzen und das Cookie mit diesem Layout aktualisieren . Wenn ein CSS-Pfad über die Eigenschaft „CssPath“ angegeben wurde, werden alle darin befindlichen Dateien eingelesen und in die ListItem Collection „<Control>.Items“ geladen.

(siehe Listing 5)

Listing 5: Die überschriebene *OnLoad*-Methode

```
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    // Cookie auslesen und Wert übernehmen
    HttpCookie cookie = this.Page.Request.Cookies["styleSwitcher_" +
this._applicationName];
    if (cookie != null)
        this._currentStyle = cookie.Value;
    // QueryString auslesen und Wert ggf. übernehmen (überschreibt das
Cookie)
    // Aktueller Wert ins Cookie schreiben
    string style = this.Page.Request.QueryString["style"];
    if (style != null) {
        this._currentStyle = style;
        HttpCookie c = new HttpCookie("styleSwitcher_" +
this._applicationName, style);
        this.Page.Response.Cookies.Add(c);
    }

    if (this._cssPath != null) {
        foreach (string path in
Directory.GetFiles(HttpContext.Current.Server.MapPath(this._cssPath))) {
            this.Items.Add(new ListItem(Path.GetFileName(path), path));
        }
    }
}
```

Die überschriebene OnPreRender Methode registriert das benötigte Stylesheet-Include per (im Quellcode vorhander) Hilfsmethode im Header der Seite. Zusätzlich wird die zum Control

dazugehörende Javascript-Datei registriert. Anschließend werden fünf Javascript-Variablen direkt in den Quellcode geschrieben, da sie sich häufig ändern können und ein externes Auslagern keine Cachingvorteile bringt. Die Werte der Variablen werden benötigt, um die clientseitigen Elemente zu identifizieren (<Control>.ClientID) und alle restlichen benötigten Informationen auf dem Client für Javascript zur Verfügung zu haben (siehe Listing 6).

Listing 6: Die überschriebene *OnPreRender*-Methode

```
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);

    string styleSwitcherName = "styleLink";
    string currentStyle = String.IsNullOrEmpty(this._currentStyle)
                          ? "empty.css" : this._currentStyle;

    // Aktuelle CSS Datei mit einem "link"-Tag ausgeben
    WebTools.RegisterStyleSheetIncludeToHeader(
            this.Page, styleSwitcherName,
            Path.Combine(this._basePath, currentStyle).Replace("\\", "/")
            );

    string styleClientId =
this.Page.Header.FindControl(styleSwitcherName).ClientID;

    // Benötigtes Javascript Include einfügen
    string url = this.Page.ClientScript.GetWebResourceUrl(this.GetType(),

"t4m.Controls.StyleSwitcher.js");
    WebTools.RegisterJavascriptIncludeToHeader(
        this.Page, "styleSwitcherJavascript", url);

    // Falls ein Basispfad gesetzt wurde, dieser an Javascript mitteilen
    if (this._basePath != null) {
        // Falls benötigt, ein Slash an den Basispfad anhängen
        if (!this._basePath.EndsWith("/"))
            this._basePath += "/";
    }

    WebTools.RegisterJavascriptBlockToHeader(this.Page, "initialValues",
        string.Format("var basePath = '{0}';" +
                      "var applicationName = '{1}';" +
                      "var styleLinkClientID = '{2}';" +
                      "var styleSwitcherClientID = '{3}';" +
                      "var currentStyle = '{4}';",
                      this._basePath,
                      this._applicationName,
                      styleClientId,
                      this.ClientID,
                      this._currentStyle
                      ));
}
```

Die überschriebene AddAttributesToRender-Methode schreibt das nötige „onchange“-Attribut zum Select-Feld hinzu, falls base.Render() in der untenstehenden Render-Methode aufgerufen wird. Mit anderen Worten, wenn die RenderingMode auf „DropDown“ oder „Combined“ festgelegt wird. Dieses Attribut sorgt dafür, dass beim Wechseln eines Layouts in der DropDownList die Javascript-Methode aufgerufen wird, die den Layoutwechsel vornimmt (Siehe Listing 7).

Listing7: Die überschriebene *AddAttributesToRender*-Methode

```
protected override void AddAttributesToRender(HtmlTextWriter writer) {
    writer.AddAttribute("onchange", "switchStyle(this,
this[this.selectedIndex].value);");
    base.AddAttributesToRender(writer);
}
```

In der überschriebenen Render-Methode läuft das gesamte Rendering für das Control ab. Je nachdem, welche RenderingMode aktiv ist, wird zuerst „base.Render()“ aufgerufen, um die DropDownList vor der ungeordneten Liste zu rendern.

Die DropDownList rendert sich selber anhand der gefüllten ListItem-Collection. Genau dieselbe Collection kann auch für das Rendering der ungeordneten Liste benutzt werden. Mit der im kompletten Quellcode vorhandenen Methode „GetAllQueryStringParams“ werden eventuell vorhandene QueryString-Parameter ausgelesen und an die Links angehängt. Im letzten Teil wird optional ein ListItem für das „StyleShow“-Experiment gerendert (siehe Listing 8).

Listing 8: Die überschriebene *Render*-Methode

```
protected override void Render(HtmlTextWriter writer) {
    if (this._renderingMode.Equals(RenderingMode.DropDown)
        || this._renderingMode.Equals(RenderingMode.Combined)) {
        base.Render(writer);
    }

    if (this._renderingMode.Equals(RenderingMode.List)
        || this._renderingMode.Equals(RenderingMode.Combined)) {
        writer.WriteBeginTag("ul");
        writer.WriteAttribute("id", this.ClientID + "list");
        writer.Write(">");
        foreach (ListItem style in this.Items)
        {
            writer.WriteFullBeginTag("li");
            writer.WriteBeginTag("a");

            // Falls der Style des derzeitigen Menüpunktes aktiv ist,
            // wird diesem die aktive CSS Klasse zugewiesen
            if (style.Value == this._currentStyle)
                writer.WriteAttribute("class", this._activeCssClass);

            writer.WriteAttribute("href",
                String.Format("?style={0}{1}",
                              style.Value,
                              WebTools.GetAllQueryStringParams(this.Page,
"style"))
                              );

            writer.WriteAttribute("onclick",
                                  String.Format("return switchStyle(this,
'{0}');",
                                  style.Value));
            writer.Write(">");
            writer.Write(style.Text);
            writer.WriteEndTag("a");
            writer.WriteEndTag("li");
        }

        if (this._styleShowEnabled) {
            writer.WriteFullBeginTag("li");
            writer.WriteBeginTag("a");
```

```
            writer.WriteAttribute("id", "startStyleShow");
            writer.WriteAttribute("href", "#");
            writer.WriteAttribute("onclick", "startStyleShow(0);");
            writer.Write(">");
            writer.Write("Start StyleShow");
            writer.WriteEndTag("a");
            writer.WriteEndTag("li");
        }

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

Der Javascript-Teil in Kürze

In der obenstehenden „OnPreRender"-Methode wird eine Javascript-Datei registriert, die in der Control-Assembly eingebunden[1] ist. Im folgenden Teil können Sie die benötigten Javascript-Funktionen erkunden, die für die clientseitige Funktionalität benötigt werden.

Mit der im Listing 9 ersichtlichen Javascript-Funktion wird das aktuelle Layout im korrekten Cookie gespeichert.
Innerhalb der Funktion werden unter anderem auch die in der obenstehenden OnPreRender-Methode geschriebenen Javascript-Variablen benutzt.

Listing 9: *saveStyle* Javascript-Funktion

```
function saveStyle(style) {
    var cookie = 'styleSwitcher_' + applicationName + '=';
    cookie    += style + ';';
    var exp = new Date();
    exp.setTime(exp.getTime() + (30 * 24 * 60 * 60 * 1000));

    cookie    += 'expires=' + exp.toGMTString() + '; ';
    cookie    += 'path=/;';
    document.cookie = cookie;
}
```

Mithilfe der Funktion im Listing 10 wird das Layout gewechselt. Auch hier sehen Sie wieder die Benutzung der vorher bereitgestellten Variabeln, um clientseitig alle Infos beisammen zu haben.

Listing 10: *switchStyle* Javascript-Funktion

```
function switchStyle(obj, style) {
    // Stylesheet Umschalten
    var styleLink   = document.getElementById(styleLinkClientID);
    if(styleLink != null && style.length != 0 && style != styleLink.href)
        styleLink.href = basePath + style;

    // Aktiver Style speichern
    currentStyle = style;

    // Aktiver Menüpunkt (RenderingMode.List) aktiv setzen
    if(obj != null) {
        clearAllActive();
        obj.className = "active";
    }

    // Cookie speichern
    saveStyle(style);
```

```
    // PostBack verhindern (Aufruf: "return <function>();)
    return false;
}
```

Die kleine Hilfsfunktion in Listing 11 geht alle Punkte der unteordneten Liste durch und setzt das Attribut „className“ auf ein leeres Stringliteral, um später den aktuellen Layout-Punkt hervorheben zu können.

Listing 11: *clearAllActive* Javascript-Funktion
```
function clearAllActive() {
    var styleSwicher = document.getElementById(styleSwitcherClientID);
    if(styleSwicher != null) {
        var links = styleSwicher.getElementsByTagName('a');
        for(var i=0; i < links.length; i++)
            links[i].className = '';
    }
}
```

Experiment: StyleShow

Ganz interessante Effekte können Sie dann erzielen, wenn CSS-Dateien schrittweise geladen werden. Dazu braucht es einen HttpHandler, der aufgrund eines „step“-Parameters immer einen größeren Teil der aktiven CSS-Datei zum Browser liefert. Dabei wird das komplette Design langsam und schrittweise im Browser aufgebaut. Den Quellcode des StyleShow-Handlers ist im kompletten Code mit dabei.
Die dazugehörige Javascript-Funktion sehen Sie im Listing 12.

Listing 12: *startStyleShow* Javascript-Funktion
```
function startStyleShow(step) {
    var obj = document.getElementById(styleLinkClientID);
    var info = document.getElementById('startStyleShow');
    obj.href = basePath + 'StyleShowHandler.axd?style=' + basePath +
currentStyle + '&step=' + step.toString();
    info.innerHTML = (20 - step).toString();

    if(step != 19) {
        step++;
        window.setTimeout('startStyleShow(' + step.toString() + ');',
1500);
    } else {
        info.innerHTML = 'Start StyleShow';
    }
}
```

Fazit

Sie haben anhand einer Beispielimplementation gesehen, dass mit der Kombination von ASP.NET, Javascript und CSS vieles möglich ist.
Dabei ist ein sehr wichtiger Punkt, das es trotz Komfort-Optionen per Javascript möglich ist, ein Control zu schreiben, das auch ohne Javascript zu 100% funktionsfähig bleibt.

Der Autor

Peter Bucher ist hauptberuflich als Webentwickler tätig und wurde von Microsoft als MVP für ASP / ASP.NET ausgezeichnet. Neben der Arbeit interessiert Peter sich zusätzlich für die Spieleentwicklung

mit .NET. Sie können ihn über sein Weblog unter http://www.aspnetzone.de/blogs/peterbucher/ erreichen oder direkt eine E-Mail an peter.bucher@aspnetzone.de schicken.

Referenzen

- [1] http://www.aspnetzone.de/blogs/peterbucher/archive/2007/07/15/ressourcen-in-customcontrol-assembly-einbetten-und-benutzen.aspx
- [2] http://www.peterbucher.ch/