Visual Web Developer

Franz Fiala

Dieser Artikel ist eine Einladung an alle, die ein Web verwalten, das kostenlose Tool Visual Web Developer 2007 Express zu verwenden. Auch alle "Nur-Html-Programmierer" kommen in den Genuss der Unterstützung eines professionellen Tools.

Der Visual Web Developer ist ein Werkzeug für Entwickler von Web-Anwendungen. Er ist spezialisiert auf ASP.NET und das DotNet-Framework. Er kann auch mit einfachen Html-Seiten umgehen aber das ist nicht sein Hauptaufgabengebiet. Seine Stärke ist die Generierung Html-Seiten ohne Verstöße gegen die Entwurfsregeln von Html, immer auch optimiert für den jeweils verwendeten Browser ohne, dass man sich als Entwickler um diese Details kümmern muss.

Um die grundsätzliche Vorgangsweise und die wichtigsten Features zu zeigen, werden wir eine Musterseite und danach eine Muster-Website entwerfen, fast ohne Programm. Diese Beschreibung der ersten Schritte mit dem Visual Web Developer soll den Einstieg erleichtern und die Entscheidung erlauben, ob man dieses Tool für zukünftige Arbeiten an Internet-Projekten in die engere Wahl zieht. Projekte mit mehr Programmanteil werden in den PCNEWS immer wieder vorgestellt.

Web-Designer

Jeder, der Web-Design betreibt, wird nicht nur beim Entwurf einer Seite sondern bei jeder Datenänderung gefordert. Seine Arbeit ist das Verbinden von Daten mit einem Layout. Ändern sich die Daten, muss er im HTML-Kode Änderungen vornehmen. Und spätestens nach der zweiten händisch kodierten Html-Tabelle fragt man sich, wie man diese arbeitsintensiven Vorgänge flexibler gestalten kann.

Diese unbefriedigende Arbeitsweise führte zu Lösungen auf verschiedenen Ebenen. Einerseits wurde das direkte Formatieren von Texten durch zentralisierte Style-Sheets von Formatieranweisungen entlastet.

Web-Developer

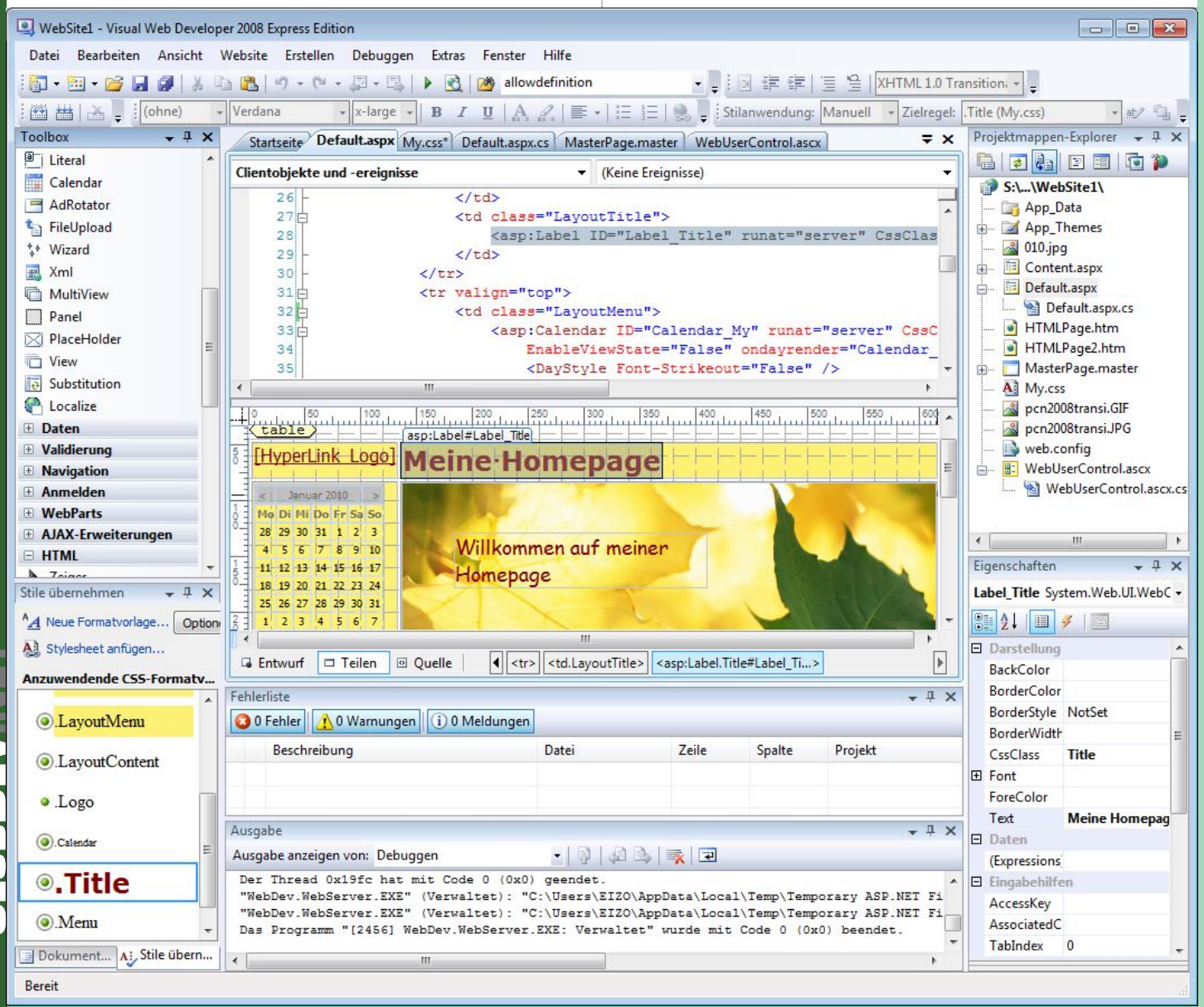
Dynamische Websites ersetzen die Handarbeit des Web-Designers durch Programme, die Daten in Datenbanken mit dem Layout verbinden und bei Anforderung durch den Benutzer generieren. Der Web-Developer stellt diese Programme her. Erst diese Dynamik bereitete die Grundlage für "Web2.0".

Der Visual Web Developer trennt Design (ASPX, ASCX, MASTER-Dateien) und Programmkode (CS– und VB-Dateien).

Diese Aufgabenstellungen müssen nicht allein den Profis vorbehalten sein; mit ein bisschen Starthilfe sollte es auch für Amateure machbar sein, professionelle Webseiten zu entwerfen. Der wichtigste Behelf ist eine Entwicklungsumgebung bestehend aus Informationen und geeigneten Tools.

Websprachen

Die ersten Programmiersprachen waren die Skriptsprachen Perl, PHP und ASP. Diese damit produzierten Programmkodes sind eine sehr unübersichtliche Mischung aus Formatierelementen und Daten. Der Kode



ist schwer lesbar, Änderungen am Layout sind nur im Kode aber nicht in einem Entwurfswerkzeug möglich.

Microsofts Antwort auf dieses Problem war ein Programmkonzept, bei dem eine klare Trennung zwischen Programm, und Darstellung ermöglicht wird. Dieses Konzept besteht aus mehreren Elementen:

- der XML-Entwurfssprache ASP.NET, die Grundlage für den zur Laufzeit generierten Html-Kode ist.
- einer objektorientierten Programmiersprache (Visual Basic oder C#)
- einer Klassenbibliothek DotNet-Framework (Aktuell Version 3.5)

Diese Programmiertechnik ist ein Quantensprung im Vergleich mit den früheren interpretierenden Skriptsprachen ASP und PHP. Statt prozeduraler Programmierung, erfolgt jetzt eine objektorientierte Programmierung. Zu vergleichen etwa mit dem Übergang vom ehemaligen Spaghetti-Kode zu prozeduraler Programmierung.

Die Veränderungen sind so weitgehend, dass bestehende ASP-Programme gänzlich neu geschrieben werden müssen. Es gibt zwar einen ASP-Kompatibilitätsmodus, den man aber nur behelfsmäßig und in einfachen Fällen einsetzen kann.

Die Grundlage für den generierten HTML-Kode ist ASP.NET, eine Vorstufe für den eigentlichen HTML-Kode im XML-Format. Anders als HTML unterliegt ASP.NET einem strengen Syntaxcheck; Abweichungen sind nicht erlaubt; aber keine Angst, man kann jederzeit ASP.NET und HTML in einer Datei mischen; das wird man aber nur in einfachen Fällen machen, um sich nicht der Vorteile der automatischen Kode-Generierung zu berauben. Im Beispielprojekt zu diesem Artikel wird einfaches Html bei der Layout-Tabelle angewendet.

Die Programme werden kompiliert. Beim ersten Aufruf einer Seite oder bei Änderungen an der Seite wird die Seite kompiliert und das Programm in dll-Dateien gespeichert. Bei jedem folgenden Aufruf werden nur mehr die dll-Dateien aufgerufen. Die Sprachen und die Klassenbibliothek sind objektorientiert. Mehrere Tausend vorgefertigte Klassen sorgen dafür, dass man selbst nicht in die Verlegenheit kommt, eigene Klassen entwerfen zu müssen - wenigstens nicht am Anfang.

Für nicht-kommerzielle Programmierer stellt sich die Frage nach den Kosten. Aber egal, ob man sich in der Microsoft- oder Linux-Welt bewegt, die wesentlichen Hilfsmittel sind kostenlos. Da unsere Clubserver alle mit *Windows Server* ausgerüstet sind, beschränken wir unseren heutigen Ausflug auf die kostenlosen Microsoft-Tools.

Der Visual Web Developer ist Kode-orientiert. Es gibt zwar eine Design-Ansicht und zahlreiche Wizzards, die die Einstellung von Attributen erleichtern aber das Hauptaugenmerk liegt am dadurch generierten Kode. Jede Änderung am Kode kann unmittelbar in der Entwurfsansicht kontrolliert werden und umgekehrt. Das Aussehen wird durch Designs, Skins, Masterpages und Style-Sheets gesteuert.

Das Layout des Visual Web Developer

Im Bild sieht man den Web Developer in Aktion in der Entwurfsansicht. Geladen ist das Beispielprojekt für diesen Artikel, die Titelseite `default.aspx`, geteilt in Kodeansicht oben und Entwurfsansicht unten.

In der Entwurfsansicht sieht man normalerweise die Seite wie sie später im Browser ausschaut (abgesehen von Rahmen, und Hilfslinien, die als Editierhilfe dienen).

Links von der Seitenansicht sieht man die Toolbox und die CSS-Formatvorlagen und rechts den Projektmappen-Explorer mit den Dateien sowie die Eigenschaftsseite für das gerade angeklickte Objekt **`Label_Title`**.

Die Titelseite zeigt, dass man Objekte präzise positionieren kann, unterstützt durch Lineale und durch ein Raster, an dem die Objekte ausgerichtet werden können. Die Karteikarten am oberen Bildrand markieren weitere geöffnete Dokumente, die Html-Flächen am unteren Bildrand sind Links zu den HTML-Tags, die hierarchisch über dem gerade ausgewählten Element **`Label_Title`** liegen.

Toolbox

Alle Elemente der Toolbox sind nur sichtbar, wenn gerade eine ASPX– oder ASCX-Seite angezeigt wird; bei HTML-Seiten sieht man nur die HTML-Objekte, bei Programmkode-Seiten oder während des Debuggens ist die Toolbox leer.

Objekte werden aus der Toolbox auf die Entwurfsseite gezogen und dabei absolut positioniert. Die absolute Positionierung kann über *Extras -> Optionen -> HTML-Designer -> CSS-Stile* abgeschaltet werden.

Jedes Objekt wird in einem Kode-Abschnitt der Form **`<asp:ObjektTyp>.</asp:ObjektTyp>`** kodiert und hat zumindest die Attribute **`ID="ObjektTyp1"`** und **`runat="server"`**. Die **`ID`** benötigt man, um per Programm auf diese Objekt zugreifen zu können, das Attribut **`runat`** stellt sicher, dass das Objekt zur Laufzeit existiert. Bei der Einfügung eines Objekts bekommt die **`ID`** einen automatisch generierten Namen bestehend aus dem Klassennamen und einer fortlaufenden Nummer. Nach der Einfügung eines Objekts sollte man die **`ID`** umbenennen. Im Beispielprojekt wurde folgende Systematik angewendet: der Objektname bleibt gleich, angehängt wird ein Unterstrich, gefolgt von einer sprechenden Bezeichnung. Beispiele: **`TreeView_Menu`**, **`Label_Title`**.

Die wichtigsten Objekte in der Toolbox sind die Websteuerelemente im Abschnitt *"Standard"*. Alle anderen Abschnitte sind weitergehende Funktionen für *Datenbanken, Validierung, Navigation, Anmeldung* und andere.

Eigenschaften (Properties)

Die Eigenschaften eines Objekts werden im Eigenschaftsfenster rechts angezeigt. (Am Kopf dieses Fensters können die Objekte umgeschaltet werden). Für komplexe Eigenschaften gibt es weitergehende Wizzards, zum Beispiel für Listenelemente, Tabellen oder GridViews.

Stile-Verwaltung

Für Stile gibt es gleich mehrere überlappende Fenster: *"CSS-Eigenschaften"*, *"Dokumentgliederung"*, die bei Ansicht eines Style-Sheet-Dokuments auf *"CSS-Gliederung"* umschaltet, *"CSS-Formatvorlagen"* sowie das im Bild dargestellte *"Stile übernehmen"*.

Klickt man in der Entwurfsansicht auf irgendein Objekt, sieht man im Eigenschafts-Fenster rechts auch die Eigenschaft **`CssClass`**. Im Menü *"Stile übernehmen"* wird diese Klasse mit einem Rahmen markiert und ein Klick auf den rechten Rand dieses Rahmens erlaubt eine Bearbeitung mit *"Formatvorlage ändern"*. Alle Css-Eigenschaften können über ein übersichtliches Menü eingestellt werden.

Syntax-Highlighting

Selbstverständlich werden alle Elemente durch *Syntax-Highlighting* besser lesbar gemacht. Elemente, die der eingestellten Kodierungsvorschrift nicht entsprechen, werden - ähnlich wie bei einem Texteditor - durch eine Wellenlinie angemerkt. Wenn ein ASP.NET-Objekt falsch formatiert ist (fehlende schließende Klammer, fehlendes Anführungszeichen bei einem falschen Attribut), wird das Objekt in der Entwurfsansicht nicht angezeigt, es erfolgt eine Fehlermeldung. Strenge Sitten bei XML-Kode! Das Verhalten entspricht echtem Programmkode, der auch nicht kompiliert werden kann, solange ein Syntaxfehler vorliegt.

IntelliSense

Wie hieß doch das Attribut? Und schon wird ein dicker Wälzer konsultiert, um ein Detail aus Html, Css, Javascript oder C# zu finden. Nicht mit IntelliSense! IntelliSense ist ein ständig verfügbares Handbuch der gerade verwendeten Sprache. Es zeigt zu einem konkreten Punkt die verfügbare Elemente an, wenn eines ausgewählt wird, dann auch dessen Attribute oder Parametertypen aber auch deren mögliche Parameterwerte, wenn es eine Werteliste gibt, zum Beispiel bei der Farbauswahl oder bei den Ausrichtungen.

Beispielsweise zeigt der Editor beim Eintippen des Anfangs eines Tags **`"<"`** alle an dieser Stelle erlaubten Eingaben in einem PopUp-Fenster und bei Auswahl des Tags und Eingabe eines Blank, z.B. **`"<P "`** alle Attribute, die bei diesem HTML-Tag zulässig sind, an.

Aber nicht nur, bei Html, auch bei Css, JavaScript und das auch innerhalb eines JavaScript-Abschnitts. Wird das öffnende Tag geschlossen **`"<p>"`**, wird automatisch das schließende Tag **`"</p>"`** angehängt. Mit Strg J kann man diese Anzeige für jedes Element aktivieren auch wenn man nicht gerade etwas eingibt.

IntelliSense ist eine hervorragende Hilfe zum Erlernen einer diese Sprachen und beschleunigt das Arbeiten für Profis.

Kodeformatierung

Mit *Bearbeiten -> Dokument formatieren*, oder Strg K Strg D wird ein Dokument automatisch formatiert, man muss sich daher um Einrückungen nicht kümmern und hat dennoch ein übersichtliches Aussehen des Kodes.

Gliederung

Eine tolle Sache ist auch die Möglichkeit, ein Dokument zu gliedern, mit *Bearbeiten -> Gliedern -> Alle Gliederungen umschalten*, Strg M Strg L speziell bei großen Dokumenten. Dabei werden alle Kode-Teile bis auf das einschließende Tag-Paar ausgeblendet. Den gerade bearbeiteten Abschnitt kann man expandieren und in diesem übersichtlichen Bereich arbeiten.

Struktur von ASP.NET-Seitentypen

Html-Datei (HTMLPage.htm)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>



</head>
<body>
    <form id="form1" runat="server">
    <div>


-


    </div>
    </form>
</body>
</html>
```

Aspx-Datei (DefaultSingle.aspx)

```
<%@ Page
  Language="C#" AutoEventWireup="true"
  CodeFile="Default.aspx.cs"
  Inherits="Default" %>

<%@ Register Src="~/WebUserControl.ascx"
TagPrefix="My" TagName="Test" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>



</head>
<body>
    <form id="form1" runat="server">
    <div>
        <My:Test runat="server"
           ID="Test_1" />


    </div>
    </form>
</body>
</html>
```

Master-Datei (MasterPage.master)

```
<%@ Master
  Language="C#" AutoEventWireup="true"
  CodeFile="MasterPage.master.cs"
  Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <asp:ContentPlaceHolder
      id="ContentPlaceHolder_head"
      runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder
          id="ContentPlaceHolder_body"
          runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

Html-Datei

Das Html-Gerüst ist einfach und dient hier eigentlich nur zum Vergleich. Der Visual Web Developer arbeitet mit XHTML. Das sieht man an der *Document Type Definition* vor dem eigentlichen Html-Kode und am `xmlns`-Attribut im `html`-Tag. XHTML ist kurz gesagt die XML-kompatible Version von HTML. Die äußerlichen Unterschiede sind:

- Generelle Kleinschreibung der Tags und der Attribute
- Leere Elemente werden im Tag geschlossen: `<br/><hr/>`...
- Alle Attribute werden mit Anführungszeichen eingeklammert

Und andere, siehe: *http://de.selfhtml.org/html/xhtml/unterschiede.htm*

In eine Html-Datei können nur die bekannten Html-Objekte eingefügt werden (in der Toolbox des Visual Web Developers „HTML“)

Wenn nun eine Html-Datei ein Formular enthält und dessen Daten serverseitig zur Auswertung kommen sollen, muss am Server eine Programmdatei diese Daten abarbeiten. Beim Konzept von ASP.NET ist diese Datei identisch mit der Html-Datei selbst, d.h. die Eingabe des Benutzers wird im selben Kontext ausgewertet.

Aspx-Datei

Aspx steht für *Active Server Pages*. Eine Aspx-Datei kann alle Elemente einer Html-Datei enthalten aber auch alle Webserver-Steuerelemente aus ASP.NET, die alle mit `<asp:` beginnen (*http://msdn.microsoft.com/de-de/library/7698y1f0(VS.80).aspx*). Der Compiler wird im Kopf über das Schlüsselwort `Page` informiert, dass es sich um eine Seite handelt.

Zu einer Aspx-Datei gehört immer auch eine gleichnamige Kode-Datei mit der Endung C# (oder VB). Im Kopf wird mitgeteilt, wie diese Datei und die Klasse für diese Seite heißt, die

Ascx-Datei (WebUserControl.ascx)

```
<%@ Control
  Language="C#"
  AutoEventWireup="true"
  CodeFile="WebUserControl.ascx.cs"
  Inherits="WebUserControl" %>
Inhalt des Web-Benutzersteuerelements
```

Aspx-Datei (Content.aspx)

```
<%@ Page
  Language="C#"
  MasterPageFile="~/MasterPage.master"
  Title="Inhaltsseite 1" %>

<asp:Content
  ID="Content_head" ContentPlaceHolderID=
    "ContentPlaceHolder_head"
  Runat="Server">
     Meta-Tags, JavaScripts oder CSS
</asp:Content>
<asp:Content
  ID="Content_main" ContentPlaceHolderID=
    "ContentPlaceHolder_main"
  Runat="Server" >
     Der eigentliche Seiteninhalt
</asp:Content>
```

sie enthält. Diese Kodedatei ist anfangs leer und wird nur bei Bedarf verwendet.

Der gesamte Html/ASP.NET-Kode befindet sich zwischen `form`-Tags. Jede Seite übergibt alle Parameterwerte über dieses `form`-Tag per `POST` zum Server. Der Zustand der Seite bleibt bei diesen „Postbacks“ erhalten.

In eine Aspx-Seite können alle Objekte aus der Toolbox eingefügt werden.

In klassischen Webprogrammen speichert man gleichbleibende Seitenelemente wie Logos, Links und Menüs aber auch wiederverwendbare Gestaltungselemente in externen Dateien und verbindet sie mit `include` mit der eigentlichen Html/Asp/Php-Seite. Hierbei handelt es sich um eine rein textuelle Einfügung, nicht um eine programmatische.

Include-Anweisungen gibt es in ASP.NET nicht. Wiederverwendbare Module speichert man in Ascx-Dateien, gleichbleibende Seitenelemente in Master-Dateien.

Ascx-Datei

Wiederverwendbare Module kodiert man in so genannten Ascx-Dateien (*Active Server Control*, Web-Benutzersteuerelement). Die Verwendung muss man im Kopf der Seite mit `Register` ankündigen. Das Modul wird in der Seite wie ein ASP.NET-Modul eingefügt, allerdings nicht mit dem Präfix `asp` sondern mit einem selbst vergebenen Namen, im Beispiel `My`. (*http://msdn.microsoft.com/de-de/library/y6wb1a0e(VS.80).aspx*)

Master-Datei

Eine Website besteht im Allgemeinen aus vielen Seiten. Diese Seiten enthalten gleichbleibende Elemente (Menü, Farbgebung, Hinweise wie „Kontakt“, „Impressum“ und andere). Es wäre sehr arbeitsintensiv, diese Elemente auf allen Seiten bearbeiten zu müssen. Gemeinsamkeiten mehrerer Seiten kodiert man in MasterPages. Ihr Kennzeichen im Seitenkopf ist `Master`. Im Kodeteil gibt es Platzhalter (im Beispiel `ContentPlaceHolder_head` und `ContentPlaceHolder_body`). Diese Platzhalter werden zur Laufzeit mit dem variablen Kodeteil der jeweiligen Seite gefüllt. Diese Seiten sind wieder gewöhnliche Aspx-Seiten aber mit dem Zusatz `MasterPageFile` im Kopf. Der Content selbst steht in einem `Content`-Tag.

Master-Pages

http://msdn.microsoft.com/de-de/library/4xh7yfby(VS.80).aspx

Designs

http://msdn.microsoft.com/de-de/library/705sff8d(VS.80).aspx

Beispielseite

Um die grundlegenden Eigenschaften des Visual Web Developer zeigen zu können, wurde in einem Beispielprojekt eine ASPX-MusterWebsite hergestellt. Dieses Web vermeidet Programmkode, damit man sich zunächst auf die grundlegende Bedienung konzentrieren kann. Weiter gehende Projekte werden wir an dieser Stelle vorstellen.

Wir beginnen mit einer einzelnen Seite und verwenden dafür gleich die vom Visual Web Developer angelegte Startseite `Default.aspx`. Es wird aber bereits beim Entwurf dieser Seite darauf Rücksicht genommen, dass weitere Seiten hinzugefügt werden können, die im Aussehen dieser ersten Seite entsprechen und Objekte dieser ersten Seite übernehmen ohne, dass man diese Objekte in die weiteren Seiten kopieren müsste (Konzept der MasterPages).

Die Seiteneinteilung erfolgt über eine Tabelle mit zwei Reihen und zwei Spalten. Damit man im Browser besser sieht, bekommt die Tabelle das Attribut **`border="1"`**, später in der Website wird der Wert auf **`0`** zurückgesetzt.

Links oben	Logo mit Link auf diese Seite (auf allen folgenden Seiten gleich)
Rechts oben	Titel der Seite (ändert sich mit jeder folgenden Seite)
Links unten	Kalender und Inhalt (auf allen folgenden Seiten gleich)
Rechts unten	Bild mit darüber gelegtem Text (ändert sich mit jeder folgenden Seite)

Die Farbgebung ist für den Hintergrund Hellgelb und für die Schriftfarbe **`Maroon`** (**`=0x800000`**). Das Inhaltsverzeichnis hat zwar Links, die aber noch kein Ziel haben. Der Kalender zeigt das aktuelle Datum an, die Links stellen sich schräg, wenn man mit der Maus drüberfährt; Konfiguration in `My.css` Stile **`a`** und **`a.hover`**.

Diese Seite benötigt fünf Dateien:

`Default.aspx`	Beim Anlegen eines neuen Projekts wird immer gleichzeitig das Startdokument `Default.aspx` angelegt Wenn das Web über einen Browser ohne explizite Angabe einer Datei gestartet wird, sieht man den Inhalt von `Default.aspx`
`Default.aspx.cs`	Zu jeder neuen Aspx-Datei wird gleichzeitig auch eine gleichnamige Kode-Datei mit der zusätzlichen Endung `.cs` angelegt (`Default.aspx.cs`). Diese Kode-Datei enthält zunächst nur eine einzige und anfangs leere Funktion **`Page_Load()`**. Ob diese Funktion mit Programmkode gefüllt wird, hängt von der Funktion der Seite und deren Objekte ab. In diesem Beispiel ist sie leer.
`My.css`	Diese Datei wird über *Datei -> Neue Datei -> Stylesheet -> My.css* (oder im *Projektmappen-Explorer* über das *Kontext-Menü -> Neues Element hinzufügen*) angelegt und mit `Default.aspx` mit *Format -> Stylesheet anfügen* verbunden. In dieser CSS-Datei werden alle Formatanweisungen zusammengefasst. Direkte Formatierungen in der ASPX-Datei werden vermieden.
`WebUserControl.ascx`	Beim Entwurf der ersten Seite werden viele Elemente angelegt, die auf allen weiteren Seiten ebenfalls vorkommen. Damit diese Seite später als Masterseite weiterverwendet werden kann, werden alle spezifische Inhalte dieser Seite in

Visual Studio Express

Für das Entwickeln von Desktop- und Webprojekten stellt Microsoft die kostenlosen Tools der Express-Linie zur Verfügung. Diese Tools sind sehr mächtig und nur ganz wenige Eigenschaften des kostenpflichtigen Visual Studio fehlen; jedenfalls vermisst man diese zunächst nicht. Ein wesentlicher Unterschied der Express-Linie zu Visual Studio ist die Aufteilung des Programms in drei verschiedene Programme:

- Windows-Programmierung: Visual C# und Visual Basic.Net
- Web-Programmierung: Visual Web Developer

Alle kostenlosen Produkte der Express Linie findet man hier: *http://www.microsoft.com/express/*

Visual Web Developer Express

Die konkrete Download-Seite des Visual Web Developer Express ist *http://www.microsoft.com/express/web*. Die Produkte laden ein kurzes Programm, ca. 3 MB welches dann erst die eigentliche Installation startet. Man kann jedes Produkt einzeln oder auch mit dem Link *"All - Offline Install ISO image file"* eine ISO-Datei downloaden und alles auf einmal lokal installieren.

Sprachauswahl

Jeder Download enthält auch eine Option zur Sprachauswahl und dieser Option sollte man seine Beachtung schenken. Wählt man "Deutsch" wird einem das Programm vertraut vorkommen, leider kommen aber auch alle Fehlermeldungen in Deutsch. Versucht man jetzt im Internet, einer Fehlerursache auf den Grund zu gehen, dann bekommt man mit der Eingabe der deutschen Fehlermeldungen nur einen Bruchteil der möglichen Treffer. Es ist daher eine gute Idee, in diesem Fall auf die deutsche Version zu verzichten.

Dokumentation

Weiter muss man entscheiden, ob man die Dokumentation, die MSDN-Bibliothek am lokalen Rechner gespeichert haben will oder ob die Online-Version ausreichend ist. Auf der Seite *http://www.microsoft.com/express/downloads/* findet man auch einen Download-Link zur MSDN-Express-Bibliothek, die man für ein zügiges Arbeiten mit der Dokumentation braucht. Sonst genügt aber auch die immer verfügbare Online-Version: *http://msdn.microsoft.com/de-de/library/*

Registrierung

Wenn man Visual Web Developer installiert, kann man das Programm 30 Tage lang testen. Will man es auch nachher weiter verwenden, muss man das Produkt registrieren. Der Registrierungsvorgang erfordert einige Angaben zu seinen Interessen und eine E-Mail-Adresse.

Dateien

Der Visual Web Developer legt im Ordner `Documents` folgende Ordnerstruktur an:

```
Documents\Visual Studio 2008
   Backup Files
   Code Snippets
   Projects
   Templates
   Visualizers
   WebSites
```

Wenn man auf die Vollversion Visual Studio 2008/2010 umsteigt, können alle Projekte und Websites ohne Änderungen weiterbearbeitet werden.

Neues Projekt

Ein Projekt und eine Website werden durch zwei Projektmappendateien beschrieben: `NeuesProjekt.sln` (Text) und `NeuesProjekt.suo` (Binär). Im Ordner `Projects\NeuesProjekt`. Die eigentlichen Projektdaten werden entweder ebenfalls in diesem Ordner angelegt (Option *Projektmappenverzeichnis erstellen = nein*) oder in einem weiteren gleichnamigen Ordner `Projects\NeuesProjekt\NeuesProjekt` (Option *Projektmappenverzeichnis erstellen = ja*).

Neue Website

Eine WebSite legt die Projektmappendatei ebenfalls im Ordner `Projects\NeuesProjekt` an, die eigentlichen Projektdaten befinden sich aber im Ordner `WebSites\NeuesProjekt`.

diesem WebUserControl zusammengefasst und über eine Register-Anweisung im Kopf von Default.aspx mit diesem verbunden.

`WebUserControl.ascx.cs` Ebenso wie bei default.aspx wird auch bei ASCX-Dateien eine gleichnamige Kode-Datei mit der Endung .cs angelegt. Diese Datei enthält in diesem Beispiel keinen Kode.

Download und Installation

- Download: *http://pcnews.at/ins/pcn/1xx/11x/116/001600/_prg/PCN117Website.zip*
- Vorschau: *http://fiala.member.pcc.ac/117website/*
- Öffnen Sie den Visual Web Developer
- *Datei -> Neue Website -> ASP.NET-Website im lokalen Dateisystem* -> S:\Documents\Visual Studio 2008\WebSites\WebSite1
- Am lokalen Rechner werden jetzt zwei Ordner WebSite1 angelegt. Einer in Visual Studio 2008/Projects/WebSite1 (enthält die Projektmappendateien Website1.sln und Website1.suo) und einer in Visual Studio 2008/Websites/WebSite1 (enthält die Projektdateien, anfangs Default.aspx, Default.aspx.cs, web.config und den Ordner App_Data). Default.aspx ist das Stardokument. Wird es aufgerufen, erzeugt das Laufzeitsystem von ASP.NET daraus den zutreffenden Html-Kode indem die Eigenschaften des aufrufenden Browsers berücksichtigt werden. Die Programme aus der Programmkode-Datei Default.aspx.cs werden ausgeführt. Die Datei web.config konfiguriert die Anwendung und den Webserver. Das Verzeichnis App_Data soll alle Dateien enthalten, die dem User verborgen bleiben sollen, d.h. auch dann, wenn der User den exakten Pfad kennt, kann er die Datei nicht downloaden. In diesen Ordner kommen zum Beispiel Access-Datenbanken.
- Jetzt öffnen Sie die ZIP-Datei und ziehen alle Dateien in den Projektmappen-Explorer. Damit werden die Dateien dem Visual Web Developer bekanntgegeben und registriert. (Es ist auch möglich, die Dateien mit dem Windows-Explorer in das Verzeichnis Visual Studio 2008/Websites/WebSite1 zu kopieren. Man muss aber danach die Dateien im Projektmappen-Explorer über das Symbol *"Aktualisieren"* oder über den Kontext-Menüpunkt *"Ordner aktualisieren"* bekannt geben.
- Kompilieren Sie das Projekt über *Erstellen -> Website erstellen* oder über das gleichnamige Symbol. Sollte das Symbol nicht sichtbar sein, dann im Menü-Bereich über die rechte Maustaste die Symbolleiste *"Erstellen"* aktivieren.
- Öffnen Sie die Datei Default.aspx in der Entwurfsansicht; das Bild sollte der Abbildung in diesem Artikel entsprechen.
- Klicken Sie auf den kleinen grünen Pfeil in der Symbolleiste oder auf *Debuggen -> Debugging starten*. Ein im Visual Web Developer integrierter Webserver startet auf einem individuellen Port und zeigt jene Datei an, die zu diesem Zeitpunkt geöffnet war. Dieses Verhalten kann über *Website -> Startoptionen* geändert werden.
- Ein Fenster des Standard-Browsers öffnet sich und zeigt Default.aspx an

Default.aspx

Die Bildfläche wird klassisch mit einer Html-Tabelle mit zwei Zeilen und zwei Spalten in vier Bereiche unterteilt. Damit diese Grundstruktur besser sichtbar wird, wird sie hier ohne die in den Tabellenzellen enthaltenen ASP.NET-Objekte dargestellt:

```
<table class="LayoutTable">
    <tr>
        <td class="LayoutLogo">
          …
        </td>
        <td class="LayoutTitle">
          …
        </td>
    </tr>
    <tr>
        <td class="LayoutMenu">
          …
        </td>
        <td class="LayoutContent">
          …
        </td>
    </tr>
</table>
```

An den mit … gekennzeichneten Stellen werden die eigentlichen Inhalte als ASP.NET-Objekte eingefügt. Man sieht, dass keine direkte Formatierung der Zellen erfolgt, sondern über das class-Attribut Style-Sheets referenziert werden, die für das Aussehen verantwortlich sind. Diese Style-Sheets sind nicht in dieser Datei enthalten, sondern in der externen Datei My.css. Sie werden in jeder Seite dieses Webs über folgende Zeile bekannt gegeben:

```
<link href="My.css" rel="stylesheet" type="text/css" />
```

Über die Syntax dieser Zeile muss man sich bei Visual Web Developer keine Gedanken machen, denn man generiert sie über *Format -> Stylesheet anfügen*.

Eine Besonderheit auf dieser Seite sind die komplexen Websteuerelemente Calendar und TreeView, mit dem das Inhaltsverzeichnis aufgebaut wird.

Calendar muss nicht besonders parametriert werden; es zeigt im Grundzustand immer den aktuellen Monat an, was im Prinzip in Ordnung ist, nur werden aber alle Tage als Links ausgeführt und außerdem wird der aktuelle Tag nicht gekennzeichnet. Zwei Zeilen in der Ereignis-Funktion Calendar_Day_Render() ändern dieses Verhalten.

Das TreeView-Objekt muss mit den Informationen über das Inhaltsverzeichnis ergänzt werden. Das geschieht über TreeNode-Objekte, die im Tag <Nodes> eingefügt werden. Über das Attribut Expand kann angegeben werden, ob ein bestimmter Knoten im Grundzustand geöffnet oder geschlossen ist. Im Bespiel wurden alle Knoten geöffnet. Wirklich wirksam ist aber nur bei dem Knoten *"Hobbies"*, der auch untergeordnete Knoten enthält.

Der eigentliche Inhalt dieser Seite befindet sich aus der Sicht der Layout-Tabelle im Tabellen-Feld rechts unten. Dieser Inhalt ist in dieser Datei nicht enthalten, nur ein Verweis auf die externe Datei WebUserControl.ascx im Kopf über die Direktive Register.

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeFile="Default.aspx.cs" Inherits="Default" %>
<%@ Register Src="~/WebUserControl.ascx"
  TagPrefix="My" TagName="Test" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>My Homepage</title>
    <link href="My.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <table class="LayoutTable">
            <tr valign="top">
                <td class="LayoutLogo">
                    <asp:HyperLink ID="HyperLink_Logo"
                      runat="server" NavigateUrl="/">
                        <asp:Image ID="Image_Logo"
                          runat="server" CssClass="Logo"
                          ImageUrl=
                            "~/images/pcn2008transi.JPG" />
                    </asp:HyperLink>
                </td>
                <td class="LayoutTitle">
                    <asp:Label ID="Label_Title"
                      runat="server" CssClass="Title">
                      Meine Homepage
                    </asp:Label>
                </td>
            </tr>
            <tr valign="top">
                <td class="LayoutMenu">
                    <asp:Calendar ID="Calendar_My"
                      runat="server" CssClass="Calendar"
                      EnableViewState="False"
                      OnDayRender="Calendar_My_DayRender">
                        <DayStyle CssClass="Calendar" />
                    </asp:Calendar>
                    <asp:TreeView ID="TreeView_Menu"
                       CssClass="Menu" runat="server"
                       ImageSet="Arrows">
                        <Nodes>
                            <asp:TreeNode Expanded="True"
                              Text="Home" NavigateUrl="/" />
                            <asp:TreeNode Expanded="True"
                              Text="Hobbies"
                              NavigateUrl="/Hobbies.aspx" >
```

```
                                <asp:TreeNode Expanded="True"
                                  Text="Wandern"
                                  NavigateUrl=
                                    "/HobbiesWandern.aspx" />
                                <asp:TreeNode Expanded="True"
                                    Text="ClubComputer"
                                    NavigateUrl=
                                "/HobbiesClubComputer.aspx" />
                                 <asp:TreeNode Expanded="True"
                                    Text="Musik"
                                    NavigateUrl=
                                "/HobbiesMusik.aspx" />
                            </asp:TreeNode>
                            <asp:TreeNode Expanded="True"
                              Text="Bilder" NavigateUrl=
                              "/Bilder.aspx" />
                            <asp:TreeNode Expanded="True"
                              Text="Links" NavigateUrl=
                              "/Links.aspx" />
                        </Nodes>
                    </asp:TreeView>
                </td>
                <td class="LayoutContent">
                    <My:Test runat="server" ID="Test1" />
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

Default.aspx.cs

Diese Datei wird gleichzeitig mit `Default.aspx` angelegt und enthält anfangs nur die Datei **`Page_Load()`**. Um das Verhalten des **`Calendar`**-Objekts zu verändern, wird die Datei `Default.aspx` in den Entwurfsmodus geschaltet und das **`Calendar`** -Objekt ausgewählt. Im Eigenschaftsfenster sind dann die *"Ereignisse"* sichtbar. Das Ereignis **`DayRender`** wird immer durchlaufen, wenn einer der Tage des Monats gerendert wird. Ein Doppelklick auf dieses Ereignis legt die Ereignisfunktion **`Calendar_My_DayRender()`** inklusiver der Parameter an. Details zu allen diesen Dingen erfährt man immer aus der Online-Hilfe. Dort sind auch Beispiele zur korrekten Anwendung dieser Funktion enthalten, die als Vorlage für die eigenen Absichten dienen. Man erfährt, dass das übergebene Argument **`e`** die Eigenschaft **`Day`** enthält und diese Eigenschaft wieder hat die Eigenschaft **`IsSelectable`**, welche für den Link verantwortlich ist. Sie wird auf **`false`** eingestellt. Der heutige Tag wird über die Eigenschaft **`IsToday`** gemeldet und wenn **`IsToday`** zutrifft, wird die Hintergrundfarbe der Zelle auf hellgrün eingestellt.

```
using System;
using System.Web.UI.WebControls;
using System.Drawing;

public partial class Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {

  }
  protected void Calendar_My_DayRender(
    object sender, DayRenderEventArgs e)
  {
    e.Day.IsSelectable = false;
    if (e.Day.IsToday)
      e.Cell.BackColor = Color.LightGreen;
  }
}
```

WebUserControl.ascx

Das Web-Benutzersteuerelement `WebUserControl.ascx` enthält keinerlei Meta-Angaben wie das bei einer Html-Datei der Fall wäre sondern nur den Kode, der einzufügen ist. Im Beispiel ist es das Bild `010.jpg` und ein darüber gelegter Text *"Willkommen auf meiner Homepage"*, der mit absoluter Positionierung über das Bild gelegt wird. Diese beiden Elemente werden durch einen Abschnitt eingerahmt **`<div>.</div>`**, damit es unabhängig vom späteren Containerdokument immer in einer neuen Zeile beginnt.

Der spätere Namen im Dokument, in dem das Steuerelement verwendet wird, muss ebendort festgelegt werden. Zu dieser Datei `WebUserControl.ascx` gehört wie bei einer Aspx-Datei auch eine Kode-Datei `WebUserControl.ascx.cs`, die aber in diesem Fall nur die leere Methode **`Page_Load()`** enthält und daher hier nicht dargestellt wird.

```
<%@ Control Language="C#" AutoEventWireup="true"
    CodeFile="WebUserControl.ascx.cs"
    Inherits="WebUserControl" %>
<div>
    <img style="width: 477px; height: 340px;" src="010.jpg"
        alt="Herbstimpressionen" />
    <asp:Label ID="Label_Welcome" runat="server"
        Style="z-index: 6; left: 185px; position: absolute;
        font-family: 'Comic Sans MS'; color: #800000;
        font-size: large; width: 225px;
        margin-top: 0px; top: 130px;"
        Text="Willkommen auf meiner Homepage">
    </asp:Label>
</div>
```

My.css

Die Style-Sheets für diese Demoseite sind sehr einfach gehalten. Das **`body`**-Tag legt die grundlegenden Eigenschaften der Schrift fest. Außerdem wird hier eingestellt, dass es keinen Rand zwischen der Layout-Tabelle und dem Browser-Fenster gibt (**`margin: 0`**). Die Styles, die mit **`Layout…`** beginnen, betreffen die Layout-Tabelle, die anderen Styles die gleichnamigen Inhalts-Objekte. Das Editieren der einzelnen Styles kann über einen Wizzard mit *Stile -> Stil erstellen* erfolgen oder auch direkt im Kode, wobei IntelliSense alle möglichen Eingaben in einem Vorschaufenster angibt.

```
body
{
    background-color: White;
    color: Maroon;
    font-family: Tahoma;
    font-weight: normal;
    margin: 0;
}
.LayoutTable
{
    height: 374px;
    margin-bottom: 0px;
    width: 100%;
    height: 100%;
    background-color: #FDF276;
}
.LayoutLogo
{
    vertical-align: middle;
    width: 120px;
}
.LayoutTitle
{
    vertical-align: middle;
    height: 19px;
}
.LayoutMenu
{
    width: 120px;
    vertical-align: top;
    text-align: left;
}
.LayoutContent
{
    vertical-align: top;
}
.Logo
{
    width: 120px;
}
.Calendar
{
    font-size:x-small;
    width: 87px;
}
.Title
{
    font-family: Verdana;
    font-weight: bold;
    width: 542px;
    height: 24px;
    font-size: x-large;
}
.Menu
{
    margin-left: 20px;
}
a
{
    color: Maroon;
}
a:hover
{
    font-style: italic;
}
```

Mehrere Seiten

Eine Website besteht aus vielen Seiten, die aber durch ein gemeinsames Layout und viele gemeinsame Objekte auf allen Seiten verbunden sind. ASP.NET unterstützt ein solches Konzept durch MasterPages. Die MasterPage enthält alle gemeinsamen Elemente, jede einzelne Seite enthält nur jene Elemente, die für diese Seite allein benötigt werden.

Hier werden die Schritte beschrieben, die notwendig sind, um das mit der ersten Seite entwickelte Konzept mehrseitig zu machen. Wir lassen alle entstandenen Dateien unverändert und legen uns Kopien davon an.

My.Master

(1) Masterdatei anlegen mit

Datei -> Neue Datei -> Masterseite -> ja Kode in eigener Datei platzieren -> nein Masterseite auswählen -> `"My.master"`

Alle gemeinsamen Elemente aus `default.aspx` in die Masterdatei übertragen.

(2) Die bestehende Datei `Default.aspx` enthält praktisch alle gemeinsamen Elemente. Aus `default.aspx` entsteht die MasterPage. Dazu markierten wir die Datei `Default.aspx` im Projektmappen-Explorer und kopieren die Datei mit Strg C Str V. Es entsteht die Datei `"Kopie von Default.aspx"`, die auf `My.master` umbenannt wird. Die

gleichnamige Programmkode-Datei wird dabei automatisch mit kopiert und ebenfalls umbenannt. Wir ersetzen in der aktuelle Kopfzeile von `My.master` den Seitentyp von **Page** auf **Master** und die vererbte Klasse von **Default** auf **My**, sodass entsteht:

```
<%@ Master Language="C#" AutoEventWireup="true" Code-
File="MasterPage.master.cs" Inherits="My" %>
```

Wir ersetzen in der Klassendefintion von `My.master.cs`

```
public partial class Default : System.Web.UI.Page
```

durch

```
public partial class My : System.Web.UI.MasterPage
```

Weiters ersetzen wir die Inhalte in dem Tabellenfeld rechts oben (**LayoutTitle**), und rechts unten (**LayoutContent**) sowie im Kopfteil **<head>.</head>** das Title-Tag durch einen Platzhalter für den eigentlichen Inhalt nach folgendem Muster, wobei das **xx** ersetzt wird durch **Head**, **Title** und **Content**.

```
    <asp:ContentPlaceHolder id="ContentPlaceHolder_xx"
runat="server">
    </asp:ContentPlaceHolder>
```

Gerüst für Inhaltsseiten

Die Startseite `default.aspx` und alle Seiten, die im Menü aufgerufen werden, haben denselben Aufbau:

```
<%@ Page Language="C#" MasterPageFile="~/My.master"
        Title="Titel (erscheint im Browserkopf)" %>
<asp:Content ID="Content_Head" runat="Server"
     ContentPlaceHolderID="ContentPlaceHolder_Head">
     seitenspezifische Metatags
</asp:Content>
<asp:Content ID="Content_Title" runat="Server"
     ContentPlaceHolderID="ContentPlaceHolder_Title">
     Titel der Seite im Layout rechts oben angezeigt
</asp:Content>
<asp:Content ID="Content_Content" runat="Server"
     Inhalt der Seite im Layout rechts unten angezeigt
</asp:Content>
```

Man sieht, dass diese Seite nur jenen Kode enthält, der diese Seite von der Master-Seite unterscheidet, verpackt in so genannten Content-Server -Steuerelementen. Die grünen Stellen müssen durch den seitenspezifischen Kode ersetzt werden.

Alle anderen Inhalts-Seiten folgen diesem Muster. Die neuen Seiten müssen Namen enthalten, die bereits im Inhaltsverzeichnis festgelegt sind, d.h. wir brauchen die Seiten `Hobbies.aspx`, `HobbiesWandern.aspx`, `HobbiesClubComputer.aspx`, `HobbiesMusik.aspx`, `Bilder.aspx`, `Links.aspx` und eine neue Seite `default.aspx`.

Anlegen einer neuen Seite

Wenn man eine dieser Seiten anlegt, benutzt man folgende Kommandofolge:

Datei -> Neue Datei... -> WebForm -> Eingabe (Beispiel) `"Hobbies.aspx"` -> *Sprache "Visual C#" -> ja Kode in eigner Datei platzieren -> ja Masterseite auswählen ->* `"My.master"`

Masterseite auswählen: *ja*: es werden automatisch die drei Content-Steuerelemente angelegt, die in der Master-Datei vorgegeben sind; *nein*: es entsteht das normale Gerüst für ein ASPX-Datei wie im vorigen Beispiel angegeben.

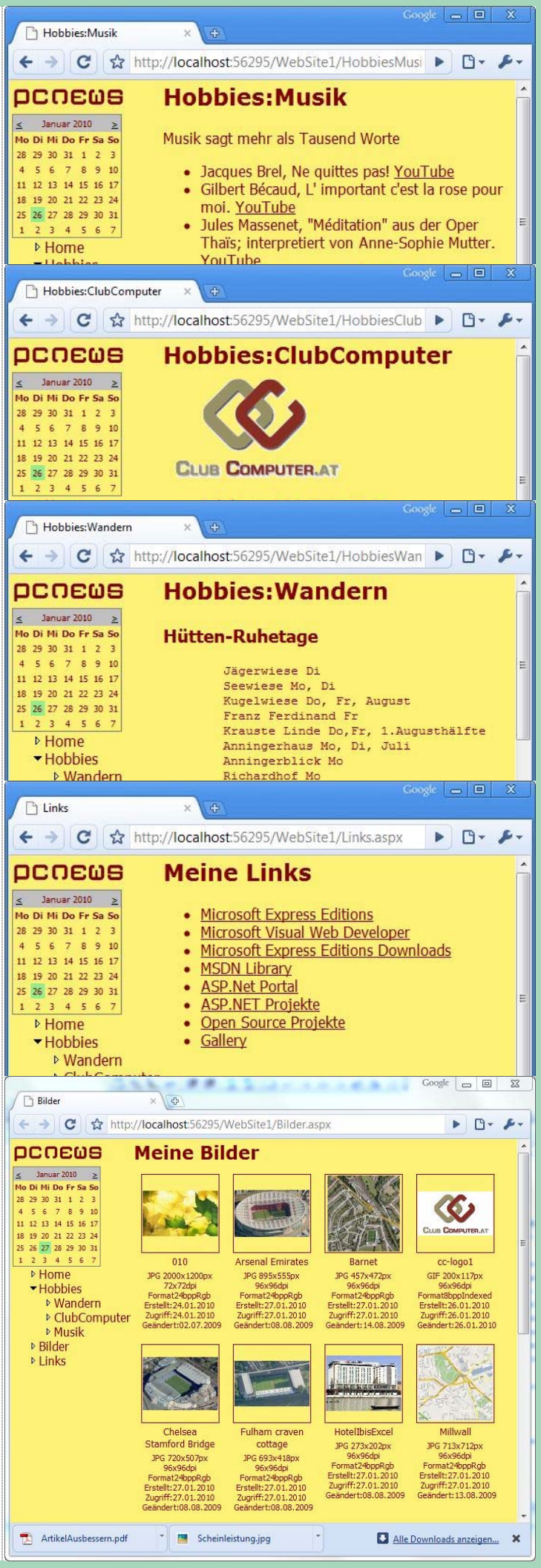
Kode in eigener Datei platzieren: *ja*: Es werden gleichzeitig zwei Dateien angelegt, die ASP.NET-Datei `Hobbies.aspx` und die Programmkode-Datei (C#) `Hobbies.aspx.cs`. Die beiden Dateien sind über die Kopfzeile in `Hobbies.aspx` miteinander verbunden; *nein*: Es wird nur eine einzige Datei `Hobbies.aspx` angelegt, die aber einen Skriptblock enthält, in den man bei Bedarf Kode hineinschreiben kann. Diese Möglichkeit kann bei kleinen Programmen angewendet werden. Ein Beispiel dafür ist das Programm Upload in PCNEWS-116, S.26. Der Vorteil ist, dass man keine weitere Datei benötigt und das Programm in jedem Verzeichnis läuft (kein virtuelles Verzeichnis nötig). Der Nachteil ist, dass man

`Default.aspx` umbenennen auf `DefaultSingle.aspx`

Das Umbenennen erfolgt im Projektmappen-Explorer über das Kontext-Menü für diese Datei. Wir ändern in `DefaultSingle.aspx` in der Kopfzeile **Inherits="Default"** in **Inherits="DefaultSingle"** und in der Programmkode-Datei `DefaultSingle.aspx.cs` den Klassennamen von **Default** auf **DefaultSingle**.

`WebUserControl.ascx` kopieren in `Default.aspx`

Das nachfolgende Muster zeigt, wie das Gerüst für eine Inhaltsseite auf das Startdokument `default.aspx` angewendet wird. Der eigentliche Inhalt wir in grün hervorgehoben. Man sieht den ursprünglichen Inhalt



von WebUserControl.ascx eingebettet in ContentPlaceHolder_Content.

```
<%@ Page Language="C#" MasterPageFile="~/My.master"
         Title="My Homepage" %>
<asp:Content ID="Content_Head" runat="Server"
     ContentPlaceHolderID="ContentPlaceHolder_Head">
</asp:Content>
<asp:Content ID="Content_Title" runat="Server"
     ContentPlaceHolderID="ContentPlaceHolder_Title">
  Meine Homepage
</asp:Content>
<asp:Content ID="Content_Content" runat="Server"
     ContentPlaceHolderID="ContentPlaceHolder_Content">
  <img style="width: 477px; height: 340px;" src="010.jpg"
     alt="Herbstimpressionen" />
  <asp:Label ID="Label_Welcome" runat="server"
     Style="z-index: 6; left: 185px; position: absolute;
     font-family: 'Comic Sans MS'; color: #800000;
     font-size: large; width: 225px;
     margin-top: 0px; top: 130px;"
     Text="Willkommen auf meiner Homepage">
  </asp:Label>
</asp:Content>
```

Dieses Rezept wird auf alle einfachen Inhaltsseiten angewendet. Einfache Texte benötigen auch keine weiteren Überlegungen und auch kein Programm. Abschließend wird an der Seite Bilder.aspx gezeigt, wie man durch ein Zusatzprogramm eine große Vereinfachung bei der Publikation von Bildern erreichen kann.

Bilder.aspx, Bilder.aspx.cs, GetImage.aspx, GetImage.aspx.cs

Einzelne Bilder bereiten beim Einbinden in eine Seite kein Problem. Wenn es aber viele Bilder sind, dann beginnt die Handarbeit langweilig zu werden. Daher wurde bei der Seite für die Bildersammlung folgendes Konzept angewendet:

- Alle anzuzeigenden Bilder (Dateiendungen jpg, jpeg, gif, png oder bmp) kommen in einen gemeinsamen Ordner, zum Beispiel images.
- Alle Bilder in diesem Ordner werden automatisch als Anordnung von Diarähmchen angezeigt
- Einstellbar sind: Bildgröße, Rahmenfarbe, Rahmenbreite, Schriftgröße für die Beschriftung, Spaltenzahl, Anzeige des Bildnamens, Anzeige von Bildinformationen
- Klickt man auf eines dieser Dias, öffnet sich ein weiteres Fenster mit einer größeren Ansicht dieses Bildes.
- Eine Besonderheit ist, dass für die Dia-Darstellung nur das kleine Vorschau-Bild vom Server gesendet wird; das Vorschaubild wird am Server aus der Originaldatei berechnet

Bilder.aspx

Die ASP.NET-Datei ist wenig spektakulär. Im Kopfteil wird eine JavaScript-Funktion definiert, die mit dem Argument pic ein neues Fenster öffnet. Die Dias selbst warden durch die Funktion/Methode Page_Load() generiert und in das Platzhalter-Steuerelement PlaceHolder_Images ein.

Bilder.aspx.cs

Die Aufgabe der Funktion Page_Load() ist das Lesen des Inhalts des Bilderordners. Was ein Bild ist, bestimmt die Funktion IsImage(). Für jedes Bild wird ein Objekt MyImage angelegt. Dieses Objekt sammelt alle Angaben zu dem Bild, die aus der Datei (Datumsangaben) und aus dem Bild (Größe, Auflösung.) entnommen werden können. Die Methode BuildImageTable() generiert eine Tabelle, bestehend aus dem Bild, dem Bildnamen und der Beschreibung. Diese kleinen Tabellen werden zu einer Anordnung von Dias mit mehreren Spalten zusammengefasst und danach in den Platzhalter PlaceHolder_Images geschrieben.

GetImage.aspx, GetImage.aspx.cs

Jedes Bild wird über die Datei GetImage.aspx in die Diarähmchen eingefügt. Diese Datei berechnet aus dem Originalbild ein kleines Vorschaubild und nur dieses Bild wird zum Client geschickt. (Was in diesem Fall fehlt, ist eine Speicherung dieser Vorschaubilder am Server, damit diese Berechnung nicht bei jedem Aufruf ausgeführt werden muss.)

Veröffentlichen

Um diese Website auf dem Club-Webspace zu veröffentlichen, muss man die Dateien uploaden. Entweder in das Wurzelverzeichnis wwwroot oder in ein Unterverzeichnis. Wenn der Upload (wie in diesem Beispiel) in ein Unterverzeichnis erfolgt, muss man dieses Verzeichnis am Server als virtuelles Verzeichnis deklarieren.

Server-Upload

Um die lokal erstellten und bearbeiteten Daten auf den Webserver zu übertragen, verwendet man die Funktion *Website -> Website kopieren.* Man bekommt eine zweispaltige Anzeige, wobei links die lokalen Inhalte und rechts der Webserver zu sehen ist. Selbstverständlich berücksichtigt das Upload-System den Bearbeitungszustand und kopiert nur jene Dateien, die seit dem letzten Upload verändert worden sind.

Für die Verbindung mit einem Web-Server gibt es vier Möglichkeiten:

Dateisystem

Diese Option kann man nutzen, wenn man das Web auf lokale Ordner oder auf freigegebene Ordner im Netz übertragen will. Das ist immer dann der Fall, wenn sich der Webserver im eigenen Netz befindet.

Lokaler IIS

Um das Web auf den lokalen Webserver localhost zu übertragen zu können, muss man den Visual Web Developer als Administrator starten.

FTP-Seite

Die klassische Übertragungstechnik funktioniert praktisch auf allen Webservern. Das Ftp-Protokoll ist aber nicht in allen Netzen erlaubt.

Remotesite

Um diese (HTTP-)Übertragungsart nutzen zu können, müssen auf dem Webserver die Frontpage-Server-Extensions installiert sein. Damit kann man aber auch in Netzen arbeiten, die Ftp nicht erlauben.

Demo-Installation

http://fiala.member.pcc.ac/117WebSite/
Zeigt das zuletzt vorgestellte System der MasterPages inklusive des kleinen Bildarchivs.

http://fiala.member.pcc.ac/117WebSite/DefaultSingle.aspx
Zeigt die einzelne Seite (erkennbar an dem Rand in der Layout-Tabelle)

http://fiala.member.pcc.ac/117WebSite/Content.aspx
Zeigt das Prinzip der MasterPages.

Wie man beim Upload der Dateien sehen kann, werden nur Programmdateien aber keine kompilierten Programme zum Server übertragen. Beim ersten Aufruf einer Datei wird das Projekt am Server automatisch kompiliert. Jede Änderung am Quellkode löst automatisch eine Neukompilierung des Projekts aus. Das Verzeichnis 117Website ist ein virtuelles Verzeichnis. Der reale Speicherort ist /PCNEWS/117/Website. Man kann das mit einer Bilddatei kontrollieren. Das Bild der Titelseite ist sowohl als *http://fiala.member.pcc.ac/117WebSite/images/010.jpg* als auch unter *http://fiala.member.pcc.ac/PCNEWS/117/WebSite/images/010.jpg* sichtbar. Programmdateien können aber an dieser zweiten Adresse nicht ausgeführt werden, weil dieses Verzeichnis nicht als Anwendung konfiguriert ist.

Zusammenfassung

Der Visual Web Developer ist für Entwicklungen auf Windows Servern unverzichtbar. Man genießt die Vorteile einer kompilierten Anwendung, kann eine Web-Anwendung wie eine Desktop-Anwendung debuggen. Der erzeugte Html-Kode wird dem jeweiligen Browser angepasst und ist entspricht dem vorgeschriebenen Standard. Mit dem Visual Web Developer arbeitet man nicht mit einer bestimmten Sprache allein. Man arbeitet mit einem Gesamtkonzept, bestehend aus Clientsprachen (JavaScript, Html), Serversprachen (C#, VB) und einer Bibliothek (DotNet-Framework), die durch eine einheitliche Oberfläche bedient werden.

Natürlich ist die hier beschriebene Arbeitsweise am Desktop sehr bequem. Man kann aber auch ganz ohne diese komplexe Oberfläche auch nur mit einem einfachen Text-Editor Programme für DotNet-Server entwickeln, denn der Server selbst verfügt ja über die Kompilierwerkzeuge wie wir beim Upload der Dateien gesehen haben.

Das DotNet-Framework existiert auch als Linux-Version (*http://www.gnu.org/software/dotgnu/*).

Ich bin froh, die Skript-Zeit von ASP/PHP hinter mir gelassen zu haben und ein Skript-Projekt nach dem anderen in die Neue Welt übertragen zu können. Ein ziemlich großes datenbankorientiertes Projekt, das mit dem Visual Web Developer erstellt wurde, ist *http://rapid.iam.at/*.