

Kapitel 44

Migration

In diesem Kapitel:

Migration von ASP zu ASP.NET	994
Migration von ASP.NET 1.x zu ASP.NET 2.0	1002
ASP.NET 2.0 zu ASP.NET 3.5	1010

Dieses Kapitel liefert Informationen zur Migration von Webanwendungen von klassischem ASP zu ASP.NET sowie von ASP.NET 1.x zu ASP.NET 2.0 und ASP.NET 2.0 (mit und ohne AJAX-Erweiterungen) zu ASP.NET 3.5.

Migration von ASP zu ASP.NET

Dass .NET-Webanwendungen »netter« werden als klassische Active Server Pages-Anwendungen, können sich alle Beteiligten erhoffen: Die Entwickler haben mehr Komfort bei Entwicklung und Debugging; die Auftraggeber freuen sich über die erhöhte Produktivität des Entwicklungsteams und auf die Anwender warten bedienungsfreundlichere und funktionsreichere Webanwendungen. So viel Nettigkeit soll aber nicht darüber hinwegtäuschen, dass der Schritt von ASP zu ASP.NET nicht in einem Tag zu schaffen ist.

Für die Migration bestehender Anwendungen von ASP zu ASP.NET gibt es verschiedene Wege. Grundsätzlich sollte aber zunächst einmal festgehalten werden, dass eine Migration nicht zwingend notwendig ist, wenn die bisherige Anwendung alle Anforderungen erfüllt. Das klassische ASP wird auch im Rahmen des .NET Frameworks und in den Internet Information Services 6.0/7.0 (Bestandteil der Windows .NET Server-Familie) nach wie vor unterstützt. Es muss also niemand befürchten, dass die ASP-Webanwendung nach Installation des .NET Frameworks oder nach einer Betriebssystemmigration nicht mehr läuft. ASP und ASP.NET sind als getrennte Internet Information Services API Extensions (ISAPI) implementiert. In der Anwendungskonfiguration der IIS kann man sehen, dass *.asp*- und *.asa*-Dateien weiterhin von *asp.dll* bedient werden, während für alle ASP.NET-Dateien (Dateierweiterung *.aspx*, *.asax*, *.asmx*, *.config*, *.vb*, *.cs*, *.js* etc.) *aspnet_isapi.dll* registriert ist.

Migrationsmöglichkeiten

ASP.NET bietet jedoch gegenüber ASP einige Vorteile, die eine Umstellung auf ASP.NET rechtfertigen könnten (siehe Kapitel 5 »Vergleiche zu anderen Oberflächentechnologien«). Dabei gibt es unterschiedliche Migrationswege, die zum Einen aus der Sicht der Gesamtanwendung, zum Anderen aus der Sicht der einzelnen Seite betrachtet werden müssen.

Migration aus Sicht der Gesamtanwendung

Aus Sicht der Gesamtanwendung gibt es drei Möglichkeiten zur Migration von ASP zu ASP.NET:

- Partielle Migration
- Vollständige Migration
- Komplette Neuentwicklung einer echten Web Forms-Anwendung

Eine vollständige Migration bedeutet, dass alle Webseiten einer Webanwendung auf ASP.NET umgestellt werden. Es besteht aber auch die Möglichkeit einer partiellen Migration – also einer Integration von ASP und ASP.NET innerhalb einer Webanwendung.

Durch die Installation von ASP.NET wird die Unterstützung für das klassische ASP nicht beeinträchtigt. ASP-Seiten (Dateierweiterung *.asp*) und ASP.NET-Seiten (Dateierweiterung *.aspx*) können innerhalb einer einzigen Webanwendung nebeneinander existieren. Sie kommen sich auch hinsichtlich der Konfiguration nicht in die Quere, weil ASP Konfigurationsinformationen in der IIS-Metabase und der Registrierungsdatenbank speichert, aber ASP.NET dazu XML-Dateien verwendet.

Allerdings gibt es keine Interoperabilität auf Objektebene, das heißt insbesondere, dass ASP und ASP.NET nicht die gleichen Session- und Application-Objektmen-gen besitzen und auf diese Weise keine Daten austauschen können. Der Austausch ist aber sehr wohl auf Basis der üblichen Möglichkeiten von HTTP (URL-Parameterliste, HTTP-Header, Cookies) möglich.

Der Entwickler kann also einzelne Seiten innerhalb einer Webanwendung auf ASP.NET migrieren, ohne damit gezwungen zu sein, alle Seiten dieser Webanwendung umzustellen. Dies ist besonders eine Alternative für einzelne Webseiten, bei denen die Ausführungsgeschwindigkeit eine entscheidende Rolle spielt. Dagegen kommt dies in der Regel nicht infrage, wenn die Zustandsverwaltung genutzt wird.

Die Umstellung einer ASP-Seite auf das Web Forms-Programmiermodell nimmt in der Regel die Ausmaße einer kompletten Neuentwicklung an. Vergleicht man eine einfache Migration unter Beibehaltung des konventionellen Programmiermodells mit der echten Web Forms-Lösung, fallen schnell zwei Dinge auf: Erstens ist die Web Forms-Lösung wesentlich eleganter und zweitens ist nicht viel von dem alten klassischen ASP-Quellcode übrig geblieben.

Man kann die Performancevorteile von ASP.NET mit überschaubarem Aufwand nutzen, indem man nur syntaktische Änderungen ausführt und auf serverseitige Steuerelemente verzichtet. Die komplette Umstellung auf das neue Programmiermodell ist aber aufwändiger und sollte sorgfältig bedacht werden. Eine bestehende Anwendung auf das neue Programmiermodell umzustellen, ist wohl nur dann ratsam, wenn ohnehin eine umfassende Änderung der Anwendung ansteht. Wenn man sich für das neue Programmiermodell entscheidet, sollte man dies auch wie eine komplette Neuentwicklung angehen. Ausschneiden und Einfügen aus der Altanwendung sollte die Ausnahme sein, um nicht wieder in alte Programmiermuster zu verfallen. Zumal Sie ja die Geschäftslogik und den Datenzugriff in größeren Anwendungen ohnehin längst in wieder verwendbare Komponenten ausgelagert haben, oder?

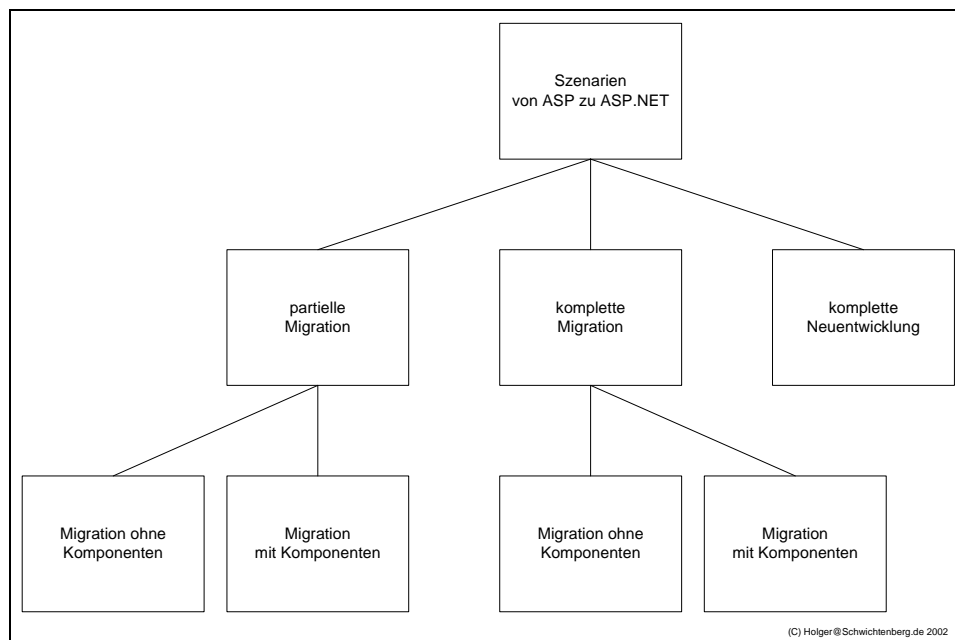


Abbildung 44.1 Migrationsszenarien

Migration der einzelnen Seiten

Der eigentliche Kern der Migrationsfrage ist der Wechsel einer einzelnen Seite von ASP zu ASP.NET. Hierbei gibt es einige Herausforderungen, die sich in drei Gebiete einteilen lassen. Das eine Gebiet sind die Änderungen an der Ablaufumgebung, also der Wechsel von dem Active Scripting Host ASP zu dem .NET Runtime Host ASP.NET. Das zweite Gebiet sind die Änderungen an der verwendeten Programmiersprache. Das dritte Gebiet ist die Umstellung des Komponentenmodells COM-Komponenten auf .NET-Komponenten. Dieser letzte Punkt soll hier nicht näher betrachtet werden, weil er (für dieses spezialisierte Buch) zu allgemeine Fragen der Migration zu .NET aufwirft.

Änderungen an der Sprache

ASP.NET unterstützt nicht mehr die COM-basierten Active Scripting-Sprachen, sondern nur noch vollwertige .NET-Sprachen. Visual Basic Script (VBScript) und JScript müssen also durch deren größere Brüder Visual Basic und JScript .NET oder die neue Sprache C# ersetzt werden.

HINWEIS Grundsätzlich ist eine händische Migration nur sinnvoll von VBScript nach Visual Basic, da die Verwendung von C# einer Neuimplementierung gleichkommt. Allenfalls sinnvoll ist eine Umsetzung von VBScript nach Visual Basic und dann eine werkzeuggestützte Konvertierung nach C#. Dieser Weg wird hier besprochen.

Es gibt zahlreiche syntaktische Unterschiede zwischen VBScript und Visual Basic. Die wichtigsten Unterschiede sind:

- Alle Let/Set-Anweisungen müssen entfernt werden.
- Alle Variablen müssen deklariert sein – wohl dem, der immer mit Option Explicit gearbeitet hat! (Dieser Zwang kann durch `<%@ Page Explicit=False %>` aufgehoben werden.)
- Alle Unterroutrinenaufrufe müssen nun die Parameter in Klammern haben. (Hier sind die im Vorteil, die immer Call verwendet haben. Call kann einfach per *Suchen/Ersetzen* entfernt werden.)
- Da Standardattribute weitgehend abgeschafft wurden, müssen nun alle Attribute explizit benannt werden. Dies gilt auch für den Zugriff auf COM-Klassen, die eigentlich eine Standardeigenschaft haben. Beispielsweise führt beim Zugriff auf eine Datenbank via klassischem ADO die Anweisung `<%=rs("ProduktName")%>` jetzt nicht mehr zur Ausgabe des gewünschten Tabellenfeldes, sondern zur Ausgabe des Objekttyps: `System.__ComObject`. Einzig richtig ist nun das, was vorher auch schon besser war: `<%=rs("ProduktName").Value%>`.

Änderungen an der Ablaufumgebung

Weitere Modifikationen des Codes sind durch Änderungen an der Ablaufumgebung ASP.NET im Vergleich zu ASP notwendig:

- Es kann nur noch eine Programmiersprache pro ASPX-Seite verwendet werden. Mehrere Sprachen können allenfalls verwendet werden, wenn Teile der Seite als User Control ausgelagert werden. Natürlich kann weiterhin jede einzelne Seite in einer anderen Sprache geschrieben sein. Unter ASP.NET stehen dafür wesentlich mehr Sprachen zur Verfügung als beim klassischen ASP.

- Während in ASP die Begrenzer `<% %>` und `<script language="CS" runat=server> ... </script>` äquivalent waren, gibt es nun einen Unterschied: Die Begrenzer `<% %>` dürfen keine Deklarationen von Unterroutinen enthalten, sondern nur noch einzelne »freistehende« Befehle. Im Umkehrschluss darf der Begrenzer `<script> </script>` nur noch Unterroutinen, aber keine freistehenden Befehle mehr abarbeiten. Dementsprechend müssen auch so genannte Renderfunktionen

```
<% Sub Footer %>
    <br><i>(C) Holger Schwichtenberg</i>
<% End Sub %>
```

umgesetzt werden in echte Unterroutinen:

```
<script language="c#" runat="server">
public void footer()
{
    Response.Write("<br><i>(C) Holger Schwichtenberg</i>");
}
</script>
```

- Das Objektmodell der Intrinsic Objects wurde erweitert. Es gibt ein zentrales Intrinsic Object Page, von dem alle anderen Intrinsic Objects abhängen. Zum Teil wurden die Mitglieder der früheren Intrinsic Objects (Response, Request, Server, Application) geändert.
So kann man nicht mehr über `Request.Forms("Feldname").Count` und `Request.Forms("Feldname")(Index)` auf ein mehrwertiges Feld zugreifen, sondern muss jetzt `Request.Forms.GetValues("Feldname").Length` bzw. `Request.Forms.GetValues("Feldname")(Index)` einsetzen. Der Index startet nun bei 0 statt bei 1. Gleiches gilt für `Request("Feldname")` und `Request.QueryString("FeldName")`. Für Felder mit genau einem Wert gibt es keine Änderung. Beim Zugriff auf Werte in Cookies ist jetzt das Attribut `Value` explizit anzugeben: `Request.Cookies("NAME").Value`. Dies gilt allerdings nicht für Cookies mit Unterwerten, die adressiert man weiterhin einfach folgendermaßen: `Response.Cookies("NAME")("UNTERNAME")`.
- Die Seitendirektive `@Language` ist zwar noch erlaubt, die empfohlene Vorgehensweise ist aber die Verwendung der neuen `@Page`-Direktive, die unter anderem ein Attribut `language` bietet. `@Page` verfügt über weitere Einstellungsmöglichkeiten und kann nicht gleichzeitig mit `@Language` verwendet werden.
- Da .NET so genannte Multiple Threaded Appartments (MTA) verwendet, muss für den Fall, dass COM-Komponenten verwendet werden sollen, die gemäß dem Single Threaded Appartment (STA)-Modell entwickelt wurden, ein spezieller Kompatibilitätsmodus aktiviert werden. Dazu bietet die `@Page`-Direktive das Attribut `aspcompat`, das so eingesetzt wird: `<%@ Page Language="CS" aspcompat="true" ...%>`. Diese Seitendirektive erlaubt auch die Verwendung des Intrinsic Objects `ObjectContext` in aufgerufenen Komponenten und damit den Zugriff auf die anderen eingebauten Objekte.
- Die Server Side Include-Anweisung wird weiterhin unterstützt, allerdings ist ASP.NET viel strenger als ASP: `<!-- #include File="funktionen1.aspx" -->` würde vom ASP.NET-Arbeitsprozess ignoriert und an den Client gesendet, weil es jeweils einen Bindestrich zu viel gibt. Richtig ist: `<!-- #include File="funktionen1.aspx" -->`.
- Ressourcen wie Datenbankverbindungen explizit zu schließen, ist unter ASP.NET wichtiger als zuvor: Durch die zeitlich nicht deterministische Speicherverwaltung in .NET könnten sonst Ressourcen unnötig lang vorgehalten werden.

Migration mit dem VWD

Der VWD bietet leider nur eine minimale Migrationshilfe für ASP-Dateien an. Beim Import einer ASP-Datei integriert Visual Studio .NET die Webseite zunächst ohne jegliche Änderung in das Projekt. Um daraus eine ASPX-Seite zu machen, ist eine manuelle Änderung der Dateierweiterung notwendig, wobei die Entwicklungsumgebung davor warnt, dass die Datei dadurch unbrauchbar werden könnte. Wenn dies dennoch bestätigt wurde, stellt der VWD fest, dass es nun eine ASPX-Datei ohne zugehörige Hintergrundcodeklassendatei gibt, und bietet freundlicherweise an, diese zu erstellen. Auch die passende @Page-Direktive wird dann eingefügt. Die Hintergrundcodeklasse erhält den Namen der ursprünglichen Seite ohne die Leerzeichen, die in Unterstriche umgewandelt werden. Der VWD legt allerdings nur einen Rumpf für die Hintergrundcodedatei an: Den Code in die passenden Ereignisbehandlungsroutinen zu übernehmen, bleibt dem Entwickler in Handarbeit überlassen.

Weiterhin bietet der VWD eine kleine Migrationshilfe von HTML-Clientsteuerelementen zu HTML-Serversteuerelementen an. Im Kontextmenü eines Clientsteuerelements findet man den Eintrag *Als Serversteuerung ausführen*. Damit wird dem Tag das Attribut `runat="Server"` hinzugefügt und gleichzeitig in der Hintergrundcodedatei ein Objekt des betreffenden Typs deklariert. Wenn man die Funktion *Details anzeigen* im VWD-Designer verwendet, kann man auch Tags wie `<Div>` und `` in HTML-Serversteuerelemente umwandeln.

Migrationsassistent

Microsoft bietet einen kostenlosen Migrationsassistenten von ASP nach ASP.NET 1.x an. Der Migrationsassistent [ASPNET01] führt einige der zuvor beschriebenen Schritte aus. Ein Migrationsassistent nach ASP.NET 2.0/3.5 existiert noch nicht.

Umlenkung der Seiten

Da sich bei einer Migration von ASP nach ASP.NET die Seitennamen ändern (von *Seite.asp* zu *Seite.aspx*), steht man bei WWW-Angeboten oft vor der Anforderung des Auftraggebers, dass die bisherige »gute« Platzierung in Suchmaschinen erhalten bleibt, also die migrierten Seiten auch weiterhin unter der alten URL erreichbar sind.

Eine Umlenkung über die *global.asax*-Seite (vgl. Kapitel 38 »Die globale Anwendungsdatei global.asax«) scheidet leider aus, denn *.asp*-Seiten lösen nicht `Application_BeginRequest()` aus. Eine Möglichkeit besteht darin, den Fehler »404 Seite nicht gefunden« durch eine Einstellung des Servers auf eine ASP.NET-Seite umzulenken und dann in dieser Seite auf die Dateierweiterung *.asp* zu prüfen. Wenn diese Dateierweiterung vorliegt, dann erfolgt statt der Ausgabe der Fehlermeldung eine Umlenkung auf die korrespondierende *.aspx*-Seite. Natürlich kann dies selektiv erfolgen. Vermeiden sollte man einen Zugriff auf Datenquellen, der viel zusätzliche Zeit verbraucht.

```
// === Umstellung von ASP auf ASPX!  
if (seite.ToLower().EndsWith(".asp"))  
{  
    seite = seite.Replace(".asp", ".aspx");  
    Response.Redirect(seite);  
}
```

Listing 44.1 Alle *.asp*-Dateien, die einen Fehler ausgelöst haben, werden auf den gleichen Seitennamen mit *.aspx* umgelenkt

Migrationsbeispiel

Die praktischen Auswirkungen der Unterschiede auf die Migration sollen an der Webanwendung *www.IT-Visions.de Buchkatalog* aus Kapitel 5 gezeigt werden. Die folgenden Listings enthalten Ausschnitte aus diesen Dateien nach der Migration (*Produktliste.aspx*, *Funktionen.incx* und *Produktdetails.aspx*). Nach diesen Änderungen ist die Webseite nun zwar mit der Dateierweiterung *.aspx* lauffähig, eine wirkliche ASP.NET-Anwendung ist es aber nicht.

Diese ASP.NET-Lösung verwendet weiterhin die COM-Komponente *ADO* für den Datenbankzugriff. Schöner wäre es noch gewesen, *ADO.NET* einzusetzen und die Variablen nicht nur zu deklarieren, sondern auch zu typisieren und damit frühes Binden einzusetzen. Dies würde aber noch mehr Änderungen erfordern; ganz bewusst soll hier zunächst die Minimallösung präsentiert werden.

Die Maximallösung der Migration, also die Neuentwicklung auf Basis serverseitiger Steuerelemente und die Umsetzung auf *ADO.NET* finden Sie in Kapitel 5.

```
<%@ Page LANGUAGE="c#" aspcompat="true" %>
<!-- #include File="funktionen.incx" -->
<HTML><body>
<%
    header;
    object rs = null;
    object farbe = null;
    object sql = null;
    object kategorie = null;
    // ----- Datenbankzugriffsobjekt (ADO) erzeugen
    rs = Server.CreateObject("ADODB.RecordSet");
    connString = ...;
    sql = ...;
    rs.open(sql, ConnString, 1, 3);
%>
    <form id="Form1" action="produktliste.aspx" method="post">
        Bitte wählen Sie einen Produkttyp:
        <select id="Select1" name="PK_ID">
            <%
while (! rs.EOF)
{Response.Write("<option value=" + rs.ToString()["PK_ID"].Value);
  if (System.Convert.ToInt32(Request["PK_ID"]) == System.Convert.ToInt32(rs("PK_ID").Value))
  {
      Response.Write(" Selected ");
  }
  Response.Write(">" + rs.ToString()["PK_Name"].Value + "</option>");
  rs.MoveNext();
}
// Recordset schließen!
rs.Close();
%>
...
rs.open(sql, ConnString, 1, 3)
%>
<table id="Table2" cellPadding="2" width="100%" bgColor="lightgrey" border="0">
...
<%
```

```
// ----- Schleife über alle Datensätze
farbe = false;
while (! rs.eof)
{
%>
...
<a href="produktdetails.aspx?P_ID=<%=rs("PPK_P_ID").value%>">
  <%=rs("P_Name").Value%>
</a>
</td>
<td align="right">
  <%=Microsoft.VisualBasic.Strings.FormatCurrency(rs("P_Preis").Value, -1,
Microsoft.VisualBasicTriState.UseDefault, Microsoft.VisualBasicTriState.UseDefault,
Microsoft.VisualBasicTriState.UseDefault)%>
</td>...
```

Listing 44.2 Ausschnitt aus *Produktliste.aspx*

Include-Datei

In der Include-Datei sind weit reichende Änderungen notwendig:

- Die Wertzuweisung an die Variable `ConnString` muss bei der Migration durch eine Funktion ersetzt werden, da in `<Script>`-Blöcken kein »freistehender« Code mehr erlaubt ist.
- Die Render-Funktion `Header()` muss komplett umgeschrieben werden, da HTML-Code innerhalb einer Unteroutine nicht mehr erlaubt ist. Alle Ausgaben müssen also mit `Response.Write()` erzeugt werden.

```
public string connString()
{
    return("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
Server.MapPath("../Daten\\ITVisions_Katalog.mdb") + ";Persist Security Info=False");
}

public void Header()
{
    Response.Write("<CENTER><TABLE id='Table1' width='100%' border='0'><TR><TD>");
    Response.Write("<P align='right'><IMG src='itv.jpg'></P></TD><TD>");
    Response.Write("<P align='left'><FONT size='6'>Produktkatalog </FONT>");
    Response.Write("</P></TD></TR></TABLE></CENTER>");
}
```

Listing 44.3 Ausschnitt aus *Funktionen.incx*

Produktdetails

Das folgende Listing zeigt die Migration der Detailseite:

```
<%@ Page LANGUAGE="c#" aspcompat="true" %>
<!-- #include File="funktionen.incx" -->
<HTML>
<body>
  <%
object rs = null;
```



```

object sql = null;
// ----- Datenbankzugriffsobjekt (ADO) erzeugen
rs = Server.CreateObject("ADODB.RecordSet");Header;
sql = "SELECT * FROM P_Produnkte where P_ID=" + Request["P_ID"];
// SQL ausführen
rs.open(sql, ConnString, 1, 3);
%>
<P>
<TABLE id="Table1" cellSpacing="1" cellPadding="1" width="100%" border="0">
  <TR>
    <TD>Produktnummer</TD>
    <TD><%=rs("P_ID").Value%></TD>
  </TR>
  <TR>
    <TD>Produktbezeichnung</TD>
    <TD>
      <%=rs("P_Name").Value%>
    </TD>
  </TR>
  <TR>
    <TD>Preis</TD>
    <TD>
      <%=Microsoft.VisualBasic.Strings.FormatCurrency(rs("P_Preis").Value, -1,
Microsoft.VisualBasic.TriState.UseDefault, Microsoft.VisualBasic.TriState.UseDefault,
Microsoft.VisualBasic.TriState.UseDefault)%>
    </TD>
  </TR>
  <TR>
    <TD>Lieferbar</TD>
    <TD>
      <%=wenn(rs("P_Lieferbar").Value,"Ja","Nein")%>
    </TD>
  </TR>
</TABLE>
<P>Sie können dieses Buch direkt beim Autor (auf Wunsch signiert) bestellen:<BR>
  <SPAN style="FONT-SIZE: 12pt">
    <SPAN style="FONT-SIZE: 12pt">
      <A href="HTTP://www.IT-Visions.de/buecher/Getbook.asp?P_ID=<%=rs("P_ID").value%>">
        Versandkostenfrei Bestellung</A>
      </SPAN></P>
    </SPAN>
  </P>
  <P>
    rs.Close();
  </P>
</body>
</HTML>

```

Listing 44.4 *Produktdetails.aspx*, die Seite mit den Buchdetails

Migration von ASP.NET 1.x zu ASP.NET 2.0

Für bestehende ASP.NET 1.x-Anwendungen gibt es im Zeitalter von ASP.NET 2.0 folgende Alternativen:

- Weiterbetreiben unter ASP.NET 1.x
- Ausführung von ASP.NET 1.x-Anwendungen unter ASP.NET 2.0 durch Änderung der Einstellung in den IIS
- Konvertierung der Anwendung nach ASP.NET 2.0

Weiterbetreiben unter ASP.NET 1.x

Grundsätzlich kann man auf einem Webserver ASP.NET-Anwendungen verschiedener Versionen betreiben. Im IIS-Manager kann bei Installation der Anwendung oder auch später im Betrieb die Version des ASP.NET Page Frameworks ausgewählt werden, die zum Ausführen der Anwendung zum Einsatz kommen soll. Die ASP.NET-Version kann über die Registerkarte *ASP.NET* für folgende Elemente im IIS-Verzeichnisbaum ausgewählt werden:

- Virtuelle Webserver
- IIS-Anwendungen

Ein virtuelles Verzeichnis kann nur dann eine eigene ASP.NET-Versionseinstellung haben, wenn es auch eine IIS-Anwendung ist.

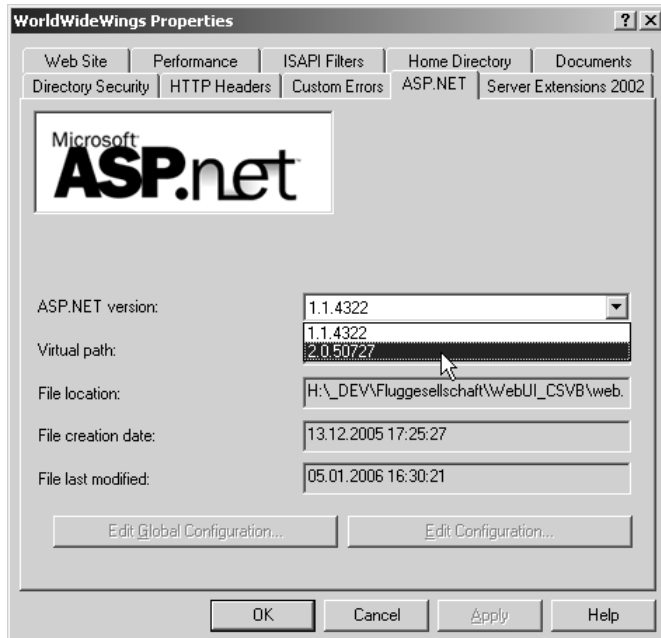


Abbildung 44.2 Auswahl der ASP.NET-Version

Grundsätzlich können auch einzelne Teile einer Webanwendung in verschiedenen ASP.NET-Versionen betrieben werden. Eine ASP.NET-Versionsfestlegung ist aber nur für IIS-Anwendungen, nicht aber für »normale« Verzeichnisse möglich.

Ausführung von 1.x unter 2.0: Einstellung in den IIS

Grundsätzlich ist es möglich, dass man eine mit ASP.NET 1.x kompilierte Anwendung unter ASP.NET 2.0 betreibt. Dazu muss man lediglich in dem IIS die ASP.NET-Version umstellen. Es gibt aber keine Garantie dafür, dass dies reibungslos funktionieren wird.

Es ist nicht möglich, innerhalb einer einzelnen ASP.NET-Anwendung mehrere CLR-Versionen gleichzeitig laufen zu lassen, das heißt Verzeichnis /x von ASP.NET 1.1 ausführen lassen und /y unter ASP.NET 2.0. Eine ASP.NET-Anwendung ist durch eine IIS-Anwendung begrenzt. Um zwei verschiedene ASP.NET-Versionen zum Einsatz zu bringen, müssten Sie in den IIS die beiden Verzeichnisse /y und /y als getrennte IIS-Anwendungen anlegen. Damit hätten Sie dann aber auch zwei getrennte ASP.NET-Anwendungen geschaffen. Der Visual Web Developer würde schon in der Anzeige die Unterverzeichnisse ausblenden (dies ist übrigens ein Trick, wie man das Ausblenden von Verzeichnissen im Websitemodell erreicht, vgl. Kapitel zum VWD). Zur Laufzeit würden die beiden Anwendungen keine gemeinsame Infrastruktur besitzen, das heißt keine gemeinsamen Sitzungen und auch keine gemeinsame Authentifizierung. Die Übergabe von Werten und Authentifizierung müsste manuell erfolgen.

TIPP

Es ist aber möglich, dass Sie in einer einzigen Webanwendung sowohl Seiten aus verschiedenen ASP.NET-Versionen als auch DLLs aus verschiedenen .NET-Versionen mischen. In diesem Fall müssen Sie die Webanwendung unter der höchsten Versionsnummer (also derzeit 2.0) betreiben und sich darüber im Klaren sein, dass auch alle älteren Seiten und DLLs von der höheren Version ausgeführt werden.

Konvertierung von ASP.NET 1.x nach ASP.NET 2.0

Um eine ASP.NET 1.x-Anwendung um neue Möglichkeiten zu erweitern, ist eine Konvertierung nach ASP.NET 2.0 bzw. von Visual Studio .NET 2002/2003 nach Visual Web Developer erforderlich.

Konvertierungsassistent

Zu beachten ist, dass Sie ASP.NET 1.x-Anwendungen nur mit Visual Studio .NET 2002/2003 bearbeiten können. Sobald Sie versuchen, die Webanwendung in Visual Studio Versionen 2005 oder 2008 zu öffnen, startet der Migrationsassistent und bietet an, die Anwendung nach ASP.NET 2.0 zu konvertieren. Sie können in Visual Studio 2005/2008 keine ASP.NET-Anwendung erzeugen oder bearbeiten, die unter ASP.NET 1.x laufen würde.

Beim Öffnen eines Webprojekts, das mit Visual Studio .NET 2002/2003 erstellt wurde, startet der VWD den *Konvertierungsassistenten* (*Visual Studio Conversion Wizard*). Während sich mit Visual Studio 2005/2008 ganze Projektmappen konvertieren lassen, kann man mit VWD Express nur einzelne Webprojekte konvertieren.

Die Konvertierung ist in der Kommandozeile folgendermaßen möglich: `devenv.exe Projektdatei /upgrade`.

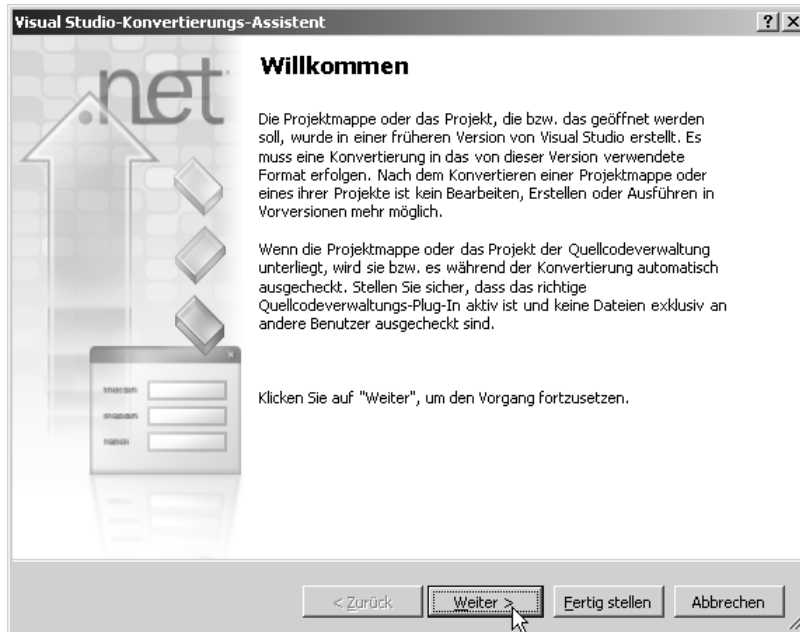


Abbildung 44.3 Startseite des Konvertierungsassistenten

HINWEIS Der Konvertierungsassistent lässt sich leider nur auf ganze Webprojekte anwenden. Eine einzelne Webseite kann nicht konvertiert werden. Zur Konvertierung einer einzelnen Webseite müssen Sie selbst Hand anlegen, indem Sie die Seitendirektive ändern und die automatisch generierten Protected-Mitglieder aus der Hintergrundcodeklasse entfernen.

Maßnahmen des Konvertierungsassistenten

Der Konvertierungsassistent nimmt bei der Migration einer Webanwendung von Visual Studio .NET 2002/2003 folgende Maßnahmen vor:

- Löschen der Projektdateien (*.vbproj*, *.csproj*) und der *.webinfo*-Datei, die in VS 2002/2003 den HTTP-Pfad zu der Webanwendung enthielt.
- Die @Page-Direktive wird geändert (CodeFile= statt CodeBehind=).
- Die Hintergrundcodeklasse erhält das partial-Schlüsselwort.
- Entfernen der automatisch generierten Protected-Mitglieder. Die von ASP.NET 1.x erstellten, aber in ASP.NET 2.0 nicht mehr benötigten Methoden InitializeComponent() und Page_Init()/OnInit() bleiben aber erhalten.
- Entfernen aller Ereignisbindungen in C#-Webprojekten, die in OnInit() mit dem Operator += implementiert wurden. Alle Ereignisbindungen außerhalb bleiben erhalten. Die Ereignisbindungen in Visual Basic-Projekten (handles-Schlüsselwort) bleiben ebenfalls erhalten.
- Verschieben aller eigenständigen Codedateien, die nicht zu einer *.aspx*-Datei gehören, in das Verzeichnis /App_Code. Dies betrifft auch die Codedatei für die *global.asax*-Datei.
- Eigenständige Ressourcendateien wandern nach /App_GlobalResources.

- Löschen der Projektdatei. Die enthaltenen Einstellungen werden entweder in der *web.config*-Datei eingetragen oder ersatzlos verworfen, weil sie nicht mehr relevant sind.
- Wenn in einer Hintergrundcodedatei neben der Hintergrundcodeklasse zusätzliche Klassendefinitionen gefunden werden, verschiebt der Migrationsassistent diese zusätzlichen Klassen in das Verzeichnis */App_Code/Migrated*.
- Wenn in einer Hintergrundcodeklasse öffentliche Mitglieder gefunden werden, erzeugt der Konvertierungsassistent dafür eine neue abstrakte Basisklasse mit dem Namen der vorherigen Seitenklasse, benennt die Hintergrundcodeklasse in *Migrated_Seitenname* um und lässt die Hintergrundcodeklasse von der abstrakten Klasse erben, um sicherzustellen, dass dieser Code von anderen Teilen der Seite noch aufgerufen werden kann.
- Alle Verweise auf andere Assemblies werden in der Datei *web.config* in Form von Einträgen im Element *Compilation/Assemblies* hinzugefügt.

Der Assistent migriert aber keine Steuerelemente, das heißt alle Steuerelemente werden belassen, auch wenn es in ASP.NET 2.0 mächtigere Steuerelemente gibt (beispielsweise *DataGrid* versus *GridView*).

TIPP

Das Ergebnis der Konvertierung wird immer an dem ursprünglichen Speicherort abgelegt und die vorherige Version überschrieben (In-Place-Konvertierung). Der Konvertierungsassistent bietet an, eine Sicherungskopie der Anwendung an einem anderen Ort zu erstellen. Nutzen Sie diese Möglichkeit!

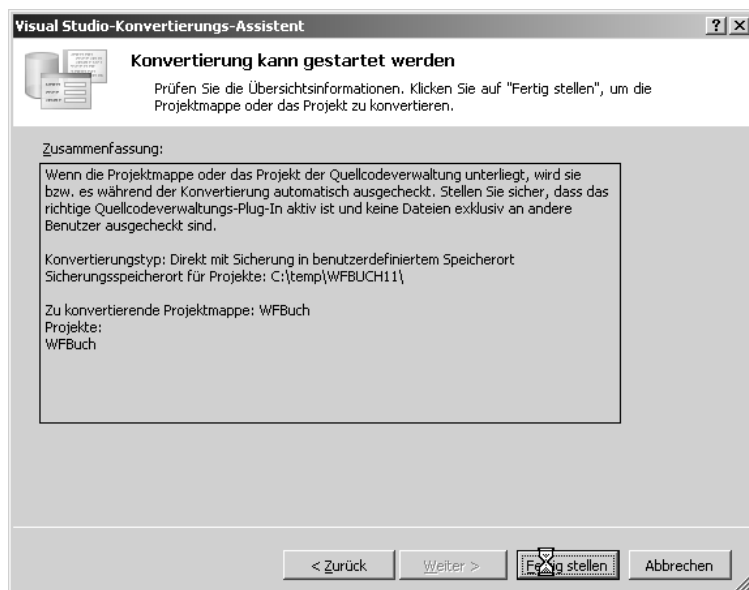


Abbildung 44.4 Zusammenfassung der vorgesehenen Arbeiten vor dem Beginn der Ausführung

Der Konvertierungsassistent legt ein Konvertierungsprotokoll an. Fast immer meldet der Konvertierungsassistent, dass es zu Fehlern gekommen ist. Das Konvertierungsprotokoll wird als Textdatei (*ConversionReport.txt*) und XML-Datei (*UpgradeLog.XML*) abgelegt. Die XML-Datei wird in Visual Studio als HTML-Datei angezeigt.



Abbildung 44.5 Abschluss des Konvertierungsassistenten

Der Konvertierungsbericht aus Abbildung 44.6 zeigt ein typisches Konvertierungsergebnis. Die Textdatei enthält mehr Informationen, da dort nicht nur Fehler und Warnungen, sondern jede Aktion des Konvertierungsassistenten vermerkt ist.

Konvertierungsbericht - WFBuch

Konvertierungsdauer: Donnerstag, 19. Januar 2006 09:39:44

Projekt: WFBuch.vbproj

Dateiname	Status	Fehler	Warnungen
WFBuch.vbproj:(Error List)		2	0
<p>Konvertierungsprobleme - WFBuch.vbproj:(Error List):</p> <p>FEHLER: Die abstrakte Basisklassendatei "controls/controls_verandern.aspx.vb" für die WF.Controls_verandern-Klasse konnte nicht erstellt werden.</p> <p>FEHLER: Die folgenden Dateien wurden nicht migriert, da sie nicht in der Projektdatdatei gefunden wurden oder die zugeordnete BuildAction auf "None" festgelegt ist. Sie müssen diese Dateien möglicherweise nach dem Konvertierungsvorgang aus dem Projekt ausschließen, um Ihre Website zu kompilieren: Dateiliste == weitere möglichkeiten\iecontrols_demo.aspx.vb,clientscripting\test.aspx,seitenmodelle\cb\cb1.aspx.vb, seitenmodelle\old\cb2.aspx.vb,datenbindung\buecherliste.aspx.vb,usercontrols\usercontrol_in_datagrid\controls\combobox.ascx,seitenmodelle\cb\cb1.aspx, clientscripting\test.aspx.vb,usercontrols\usercontrol_in_datagrid\controls\combobox.ascx.vb,usercontrols\usercontrol_in_datagrid\aktionsliste.aspx.vb, datenbindung\xml_in_datagrid.aspx.vb,datenbindung\xml_in_datagrid.aspx,usercontrols\usercontrol_in_datagrid\aktionsliste.aspx,datenbindung\buecherliste.aspx.</p>			
WFBuch.vbproj:(Warning List)		0	7
<p>Konvertierungsprobleme - WFBuch.vbproj:(Warning List):</p> <p>Warnung: In der Code-Behind-Datei "controls\Ereigniskette.aspx.vb" wurde die zusätzliche test-Klasse gefunden. Eine neue Datei "controls\Ereigniskette.aspx.vb_test.vb" wurde für diese Klasse erstellt und in den Ordner "App_Code\Migrated" verschoben. controls\Ereigniskette.aspx.vb (Line 188)</p> <p>Warnung: In der Code-Behind-Datei "datenbindung\Autorensuche.aspx.vb" wurde die zusätzliche BuchInfo-Klasse gefunden. Eine neue Datei "datenbindung\Autorensuche.aspx.vb_BuchInfo.vb" wurde für diese Klasse erstellt und in den Ordner "App_Code\Migrated" verschoben. datenbindung\Autorensuche.aspx.vb (Line 99)</p> <p>Warnung: In der Code-Behind-Datei "datenbindung\Autorensuche.aspx.vb" wurde die zusätzliche BuchListe-Klasse gefunden. Eine neue Datei "datenbindung\Autorensuche.aspx.vb_BuchListe.vb" wurde für diese Klasse erstellt und in den Ordner "App_Code\Migrated" verschoben.</p> <p>Warnung: In der Code-Behind-Datei "seitenmodelle\CB_Entwicklungszeit.aspx.vb" wurde die zusätzliche CBKomp-Klasse gefunden. Eine neue Datei "seitenmodelle\CB_Entwicklungszeit.aspx.vb_CBKomp.vb" wurde für diese Klasse erstellt und in den Ordner "App_Code\Migrated" verschoben. seitenmodelle\CB_Entwicklungszeit.aspx.vb (Line 33)</p> <p>Warnung: In der Code-Behind-Datei "usercontrols\Portal.aspx.vb" wurde die zusätzliche AnmeldeInfo-Klasse gefunden. Eine neue Datei "usercontrols\Portal.aspx.vb_AnmeldeInfo.vb" wurde für diese Klasse erstellt und in den Ordner "App_Code\Migrated" verschoben. usercontrols\Portal.aspx.vb (Line 43)</p> <p>Warnung: In der Code-Behind-Datei "usercontrols\elemente\LoginBox.ascx.vb" wurde die zusätzliche AnmeldeDaten-Klasse gefunden. Eine neue Datei "usercontrols\elemente\LoginBox.ascx.vb_AnmeldeDaten.vb" wurde für diese Klasse erstellt und in den Ordner "App_Code\Migrated" verschoben. usercontrols\elemente\LoginBox.ascx.vb (Line 76)</p> <p>Warnung: Dieses Webprojekt wurde als dateibasierte Webanwendung konvertiert. Falls Ihre Site IIS-Metadaten enthielt, z.B. als virtuelle Verzeichnisse markierte Unterordner, sollten Sie diese Website schließen und sie über den Befehl "Website öffnen" erneut öffnen. Wählen Sie dann die Registerkarte "Lokale IIS".</p>			
WFBuch.vbproj	Konvertiert	0	0
<p>Konvertierungsprobleme - WFBuch.vbproj:</p> <p>Konvertiert. Es bestehen möglicherweise ungelöste Konvertierungsprobleme, die Sie manuell beheben müssen. Weitere Informationen finden Sie unter http://go.microsoft.com/fwlink/?LinkId=46995. Sie können auch nach dem Hilfethema "Konvertieren von Visual Studio .NET 2002 oder 2003" suchen.</p>			
3 Dateien	Konvertiert: 1 Nicht konvertiert 2	2	7
<p>Konvertierungseinstellungen</p> <p>Projektmappendatei: H:_DEV\WFBuch\WFBuch.sln</p>			

Abbildung 44.6 Konvertierungsbericht für eine ASP.NET 1.1-Webanwendung

ACHTUNG Bitte denken Sie unbedingt daran, vor der Auslieferung der Webanwendung den Konvertierungsbericht an eine Stelle außerhalb der Webanwendung zu verschieben, da er sonst von Angreifern auf einfache Weise eingesehen werden kann und Aufschlüsse über Ihren Code gibt.

Herausforderungen

Bei der Migration von ASP.NET 1.x nach ASP.NET 2.0 konnte beobachtet werden:

- Der VWD unterstützt beim Websitemodell nicht mehr, dass eine Webanwendung eine andere Webanwendung referenziert, also Codeteile aus dieser aufruft. In diesem Fall müssen Sie vor der Konvertierung die relevanten Codeteile in eine eigene Klassenbibliothek auslagern. Der gegenseitige Aufruf von Seiten via HTTP ist davon nicht betroffen.

- Dateien, die in dem VS 2002/2003-Projekt nicht zur Kompilierung markiert oder ganz aus dem Projekt ausgeschlossen waren, werden bei der Migration in das Projekt integriert, da das Websitemodell automatisch alle Dateien in dem Verzeichnis als Projektbestandteil ansieht. Wenn die Dateien ausgeschlossen waren, weil sie nicht korrekt kompiliert werden konnten, werden Sie nach der Migration alle Kompilierungsfehler sehen.
- Für alle in VS 2002/2003 nicht in den Kompilierungsvorgang eingeschlossenen Dateien nimmt der Konvertierungsassistent keine Umwandlung der Seitendirektive @Page und keine Änderungen in der Hintergrundcodeklasse vor. Dies geschieht mit dem Hintergrund, dass unklar ist, warum diese Dateien ausgeschlossen sind (z.B. weil sie zu einem anderen Projekt gehören, das im gleichen Verzeichnis existiert). Der Konvertierungsassistent zeigt diese Seiten in seinem Bericht als *Fehler* an. Beim Kompilieren werden diese Dateien Fehler verursachen.
- Logischerweise kommt es zu Kompilierungsfehlern, wenn man in ASP.NET 1.x Bezeichner verwendet hat, die jetzt ein Schlüsselwort (z.B. `partial`) oder eine FCL-Klasse (z.B. `GridView`, `Profile`) darstellen.
- In ASP.NET 1.x hat der Designer für (fast) jedes Steuerelement ein `Protected`-Mitglied in die Hintergrundcodeklasse eingefügt, um das Steuerelement dort zu deklarieren. Wenn man die Sichtbarkeit dieser Mitglieder manuell von `Protected` nach `Public` geändert hat, um nach einem `Server.Transfer()` von der Folgeseite auf die Steuerelemente zuzugreifen, wird die Anwendung nicht mehr funktionieren, denn die Deklarationen werden bei der Migration entfernt und im neuen Modell automatisch im Hintergrund generiert.
- Für User Controls hat ASP.NET 1.x keine Deklarationen in die Hintergrundcodedatei eingefügt. Wer aus dem Code darauf zugreifen wollte, muss diese Deklarationen manuell anlegen. Der Migrationsassistent belässt diese Deklarationen, was zu Kompilierungsfehlern führt, denn ASP.NET 2.0 erzeugt die Deklarationen für User Controls ebenso wie für andere Steuerelemente automatisch.
- Visual Studio .NET 2002/2003 hat zu jeder ASPX-Datei auch eine Ressourcendatei (*.resx*) angelegt. Diese werden in ASP.NET 2.0 nicht mehr benötigt, sofern der Entwickler dort nicht eigene Informationen abgelegt hat. Der Konvertierungsassistent löscht die *.resx*-Dateien nicht, weil er nicht prüft, ob dort spezielle Daten untergebracht sind. Die überflüssigen *.resx*-Dateien können nach einer manuellen Prüfung gelöscht werden. Notwendige *.resx*-Dateien müssen nach *App_GlobalResources* oder *App_LocalResources* verschoben werden.
- Zu unübersichtlichem Code nach der Migration kommt man schnell, wenn man Code und Benutzerschnittstelle nicht sauber voneinander getrennt hat, insbesondere wenn man öffentlich zugänglichen Code in die Hintergrundcodeklasse verpackt hat. Der Migrationsassistent muss diesen Programmcode in das */App_Code*-Verzeichnis »auspacken«, damit er weiterhin anderen Seiten bzw. Anwendungsteilen zur Verfügung steht, die möglicherweise darauf zugreifen, denn durch die Trennung in verschiedene Assemblies wäre dies sonst nicht möglich. Oft gibt es aber gar keinen Nutzer außerhalb der Hintergrundcodeklasse für diesen Code, sondern der Code ist nur aus Unachtsamkeit öffentlich – z.B. stellt jede Methode in Visual Basic, die nicht explizit `privat` ist, eine öffentliche Methode dar. In diesem Fall erhält man von dem Konvertierungsassistenten ein zusätzliches Konstrukt mit einer abstrakten Basisklasse. Die macht die Anwendung deutlich unübersichtlicher.

- Einige Entwickler sind in ASP.NET 1.x im Rahmen des Strebens nach Wiederverwendbarkeit auf die Idee gekommen, eine andere Seitenklasse als die Basisklasse für eine andere Seitenklasse zu verwenden, Hintergrundcodeklasse *x* erbt also z.B. für Seite *x.aspx* nicht von *Page*, sondern von der Hintergrundcodeklasse *y* der Vorlagenseite *y.aspx*. Dieses Konstrukt kann in ASP.NET 2.0 nicht mehr funktionieren, weil jede Seitenklasse nur noch sich selbst und die eigenständigen Codedateien kennt. Der Migrationssassistent versucht eine Auflösung durch eine abstrakte Basisklasse wie im Fall der öffentlichen Methoden. Dabei entsteht aber weder eine syntaktisch noch semantisch korrekte Lösung. Je nach Komplexität der Vererbungshierarchie können Sie versuchen, mit der Seitendirektive *@Reference* einen Verweis zwischen den Seitenassemblies zu setzen. Dies ist auch eine Lösung, wenn ein User Control dynamisch mit *Page.LoadControl()* geladen wurde, da auch hierbei in der Regel der Objekttyp des geladenen User Controls verwendet wird.

```

' =====
' Diese Datei wurde als Teil einer ASP.NET 2.0-Webprojektkonvertierung generiert.
' Die Codedatei "App_Code\Migrated\datenbindung\Stub_buecheredit_aspx_vb.vb" wurde erstellt und enthält eine
abstrakte '-Klasse
' die als Basisklasse für die Migrated_BuecherListe-Klasse in Datei "datenbindung\buecheredit.aspx.vb"
  Syntaxfehler wird.
' Dadurch kann von allen Codedateien in dem Projekt auf die Basisklasse verwiesen werden.
' Weitere Informationen zu diesem Codemuster erhalten Sie unter http://go.microsoft.com/fwlink/?LinkId=46995.
' =====

```

Abbildung 44.7 Vom Konvertierungsassistenten verursachter Syntaxfehler

- Der VWD prüft beim Kompilieren wesentlich mehr als Visual Studio .NET 2002/2003, z.B. den Programmcode in ASPX-Dateien, *web.config*-Dateien und – optional – den HTML-Code. Dem VWD werden Fehler auffallen, die es auch unter Visual Studio .NET 2002/2003 schon gab, die aber nicht zum Tragen kamen, weil sie nur zur Laufzeit bemerkt oder diese Seiten bzw. Seitenteile selten oder nie ausgeführt wurden.
- Die Coderegion *Web Form Designer Generated Code* mit den von ASP.NET 2.0 nicht mehr benötigten Methoden *InitializeComponent()* und *Page_Init()/OnInit()* können Sie manuell entfernen, nachdem Sie sich versichert haben, dass hier kein eigener Programmcode enthalten ist.
- In C#-Projekten gibt es schwer identifizierbare Schwierigkeiten, wenn in dem alten Projekt Ereignisbehandlungsroutinen außerhalb von *InitializeComponent()* gebunden wurden. Da der Konvertierungsassistent solche Bindungen nicht entfernt und ASP.NET 2.0 die Bindungen immer automatisch generiert, kann es zu einem Doppelaufwurf der Routinen kommen, wenn das Ereignis eintritt.
- Hintergrundcodedateien können versehentlich von dem Konvertierungsassistenten nach */App_Code* verschoben werden, wenn durch einen fehlerhaften Eintrag im *CodeBehind*-Attribut einer *@Page*-Direktive die Hintergrundcodedatei nicht einer ASPX-Datei zugeordnet werden konnte.
- Zu Fehlern beim Konvertieren kann es auch kommen, wenn mehrere ASPX-Dateien auf die gleiche Hintergrundcodedatei verweisen. Dies ist in ASP.NET 1.x möglich und wurde auch für gleichartige Webseiten angewendet. In ASP.NET 2.0 ist dies nicht mehr erlaubt. Der Konvertierungsassistent wird in solchen Fällen eine Reihe von Fehlern verursachen.
- Wenn Dateien mit der Dateierweiterung *.aspx* keine *@Page*-Direktive mit dem Attributen *CodeBehind*= oder *Src*= besitzen, meldet der Konvertierungsassistent einen Fehler. In ASP.NET 1.x war es möglich, »einfachen« HTML-Seiten die Dateierweiterung *.aspx* zu geben (z.B. um sie in das Sicherheitssystem von ASP.NET einzubeziehen).

HINWEIS

Am Ende dieser Betrachtung steht eine Erkenntnis, zu der man heute oft gelangt: Wer die Standardwege geht, alle Regeln befolgt und auf vermeintlich »gute« Tricks verzichtet, ist bei einer Wartung oder Migration im Vorteil. Schwierigkeiten kriegt, wer insofern unsauber entwickelt hat, dass er Benutzeroberfläche und Code extrem unsauber miteinander vermischt hat.

ASP.NET 2.0 zu ASP.NET 3.5

Die Migration von ASP.NET 2.0 zu ASP.NET 3.5 ist recht unkompliziert, weil es keine Änderungen an dem Seitenaufbau oder den Konfigurationseinstellungen gibt, die zu Schwierigkeiten führen könnten. Die Umstellung besteht lediglich darin, einige Konfigurationseinstellungen in der Datei *web.config* ergänzen. Diese Einstellungen betreffen insbesondere die Konfigurationssektionen `<assemblies>`, `<httpHandler>`, `<system.codedom>` und die ganz neue Sektion `<webServer>`. Wie das folgende Listing zeigt, sind die notwendigen Einstellungen sehr umfangreich. Das liegt daran, dass ASP.NET 3.5 als echte Erweiterung zu ASP.NET 2.0 implementiert ist. Die bestehende *System.Web.dll* wollte Microsoft nicht ändern. Die neuen Einstellungen sorgen dafür, dass ausgewählte Funktionen auf die neue *System.Web.Extensions.dll* umgeleitet werden. In der `<assemblies>`-Sektion werden die zentralen neuen DLLs von .NET 3.5 ergänzt (insbesondere *System.Core*, *System.Xml.Linq* und *System.Web.Extensions*). Am Ende der Konfigurationsdateien werden zwei Sektionen zur Konfiguration der Compiler (`<system.codedom>`) und der IIS 7.0 (`<system.webServer>`) ergänzt.

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35">
      <sectionGroup name="scripting" type="System.Web.Configuration.ScriptingSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
        <section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>
        <sectionGroup name="webServices" type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
          <section name="jsonSerialization" type="System.Web.Configuration.ScriptingJsonSerializationSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false" allowDefinition="Everywhere"/>
          <section name="profileService" type="System.Web.Configuration.ScriptingProfileServiceSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false" allowDefinition="MachineToApplication"/>
          <section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>
          <section name="roleService" type="System.Web.Configuration.ScriptingRoleServiceSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false" allowDefinition="MachineToApplication"/>
        </sectionGroup>
      </sectionGroup>
    </sectionGroup>
  </configSections>
```

```

<system.web>
  <compilation debug="false">
    <assemblies>
      <add assembly="System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
      <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
      <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
    </assemblies>
  </compilation>

  <pages>
    <controls>
      <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      <add tagPrefix="asp" namespace="System.Web.UI.WebControls" assembly="System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    </controls>
  </pages>

  <httpHandlers>
    <remove verb="*" path="*.asmx"/>
    <add verb="*" path="*.asmx" validate="false" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    <add verb="*" path="* AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    <add verb="GET,HEAD" path="ScriptResource.axd" validate="false"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  </httpHandlers>
  <httpModules>
    <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  </httpModules>
</system.web>
<system.codedom>

  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs" type="Microsoft.CSharp.CSharpCodeProvider,System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" warningLevel="4">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" warningLevel="4">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="OptionInfer" value="true"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
  </compilers>
</system.codedom>

```

```

<system.webServer>
  <validation validateIntegratedModeConfiguration="false"/>
  <modules>
    <remove name="ScriptModule"/>
    <add name="ScriptModule" preCondition="managedHandler" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  </modules>
  <handlers>
    <remove name="WebServiceHandlerFactory-Integrated"/>
    <remove name="ScriptHandlerFactory"/>
    <remove name="ScriptHandlerFactoryAppServices"/>
    <remove name="ScriptResource"/>
    <add name="ScriptHandlerFactory" verb="*" path="*.asmx" preCondition="integratedMode"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    <add name="ScriptHandlerFactoryAppServices" verb="*" path="* AppService.axd"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    <add name="ScriptResource" verb="GET,HEAD" path="ScriptResource.axd" preCondition="integratedMode"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  </handlers>
</system.webServer>
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="System.Web.Extensions" publicKeyToken="31bf3856ad364e35"/>
      <bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity name="System.Web.Extensions.Design" publicKeyToken="31bf3856ad364e35"/>
      <bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
    </dependentAssembly>
  </assemblyBinding>
</runtime>
</configuration>

```

Listing 44.5 Konfigurationseinstellungen zum Aktivieren der neuen Funktionen in ASP.NET 3.5

Umstellungen von Projekten nach dem Websitemodell

Der VWD 2008 bietet beim Öffnen eines ASP.NET 2.0-Websiteprojekts die Umstellung auf ASP.NET 3.5 an, das heißt das Hinzufügen der Konfigurationseinstellungen (Abbildung 44.8).



Abbildung 44.8 Angebot zur Umstellung auf ASP.NET 3.5

Eine Website kann auch nachträglich durch die Einstellung *Target Framework* in den Eigenschaften des Wurzelverzeichnisses umgestellt werden. Visual Studio 2008 ergänzt bzw. ändert dann die notwendigen Konfigurationseinstellungen in der *web.config*-Datei.

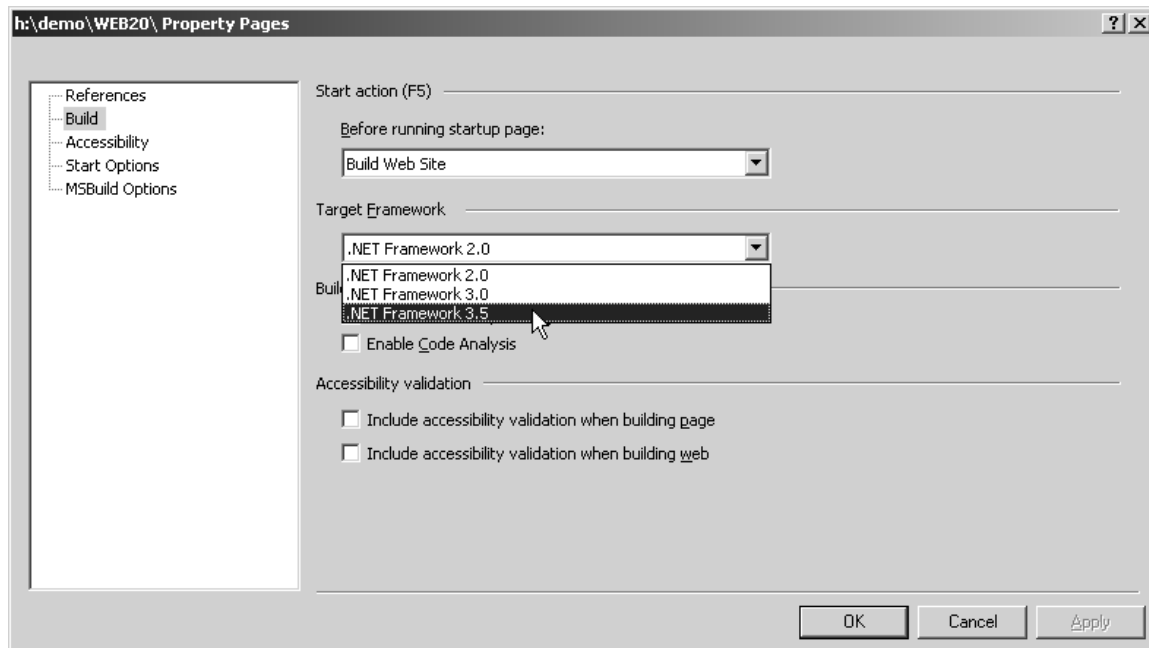


Abbildung 44.9 Einstellung des Target Framework für Website-Projekte

HINWEIS Die Umstellung von .NET Framework 2.0 zu .NET Framework 3.0 bedeutet keine Änderungen in der Konfigurationsdatei, da sich ASP.NET in .NET 3.0 nicht geändert hat. Einen Unterschied bemerkt man nur in der Elementvorlage. Erst seit .NET 3.0 gibt es WCF-Dienste.

Ein Sonderfall ist die Umstellung eines Webprojekts, das bereits die *ASP.NET AJAX Extensions 1.0* für *ASP.NET 2.0* verwendet hat. In diesem Fall existieren schon einige der Einstellungen, allerdings mit Verweis auf die ältere Version 1.0 der *System.Web.Extensions.dll*. In diesem Fall werden in der *web.config*-Datei folgende Änderungen ausgeführt:

- Verweise auf die *System.Web.Extensions* in der Version 1.0 werden gelöscht (*System.Web.Extensions 1.0* entspricht den ASP.NET 2.0 AJAX Extensions).
- Bei den ASP.NET-Handlern und ASP.NET-Modulen werden Verweise auf die *System.Web.Extensions 1.0* durch die Version 3.5 ersetzt.

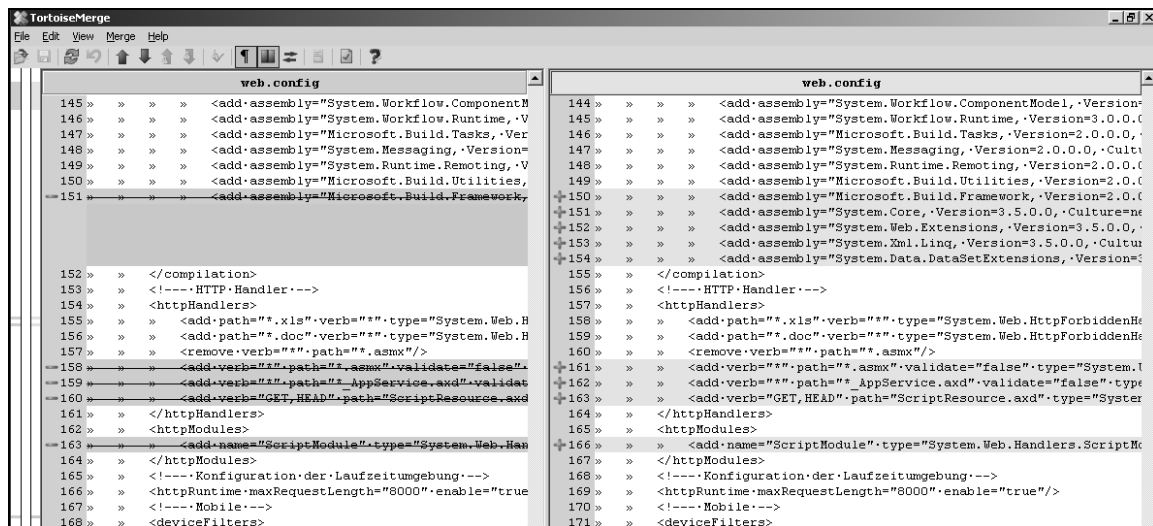


Abbildung 44.10 Ausschnitt aus den Änderungen bei der Migration von den ASP.NET AJAX Extensions 1.0 für ASP.NET 2.0 nach ASP.NET 3.5. Links die alte `web.config`-Datei, rechts die veränderte.

Umstellungen von Projekten nach dem Webanwendungsmodell

Bei Projekten, die das Webanwendungsmodell verwenden, ist das Verhalten anders. Der VWD startet den allgemeinen Projektmigrationsassistenten und stellt die Projektdatei auf das Visual Studio 2008-Format um. Wenn man den Assistenten nicht durchlaufen möchte, kann man das Projekt im VWD 2008 nicht verwenden. Der Projektmigrationsassistent nimmt zunächst gar keine Änderungen vor. Die Einstellungen in der `web.config`-Datei, die ASP.NET 3.5 benötigt, erscheinen erst nach der Umstellung des *Target Framework* in den Projekteigenschaften (Abbildung 44.11).

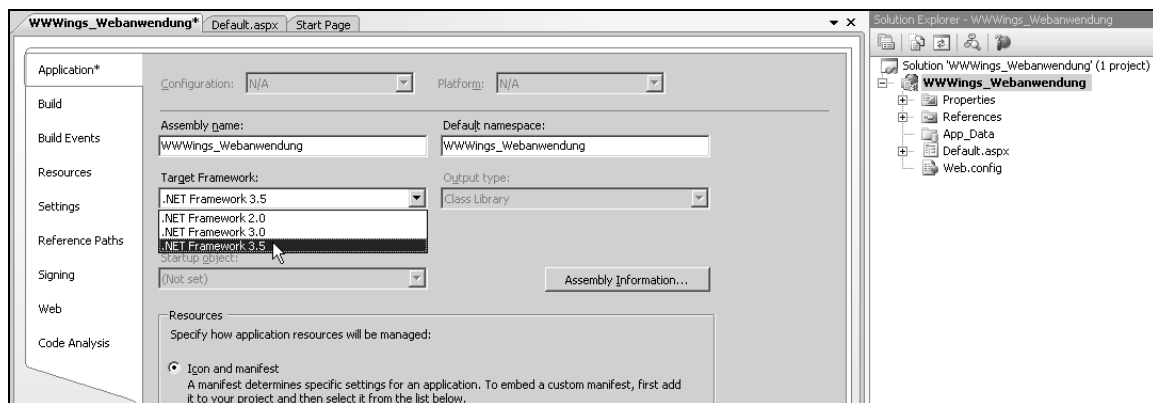


Abbildung 44.11 Einstellung bei einem Webanwendungsprojekt (bei C#-Projekten)