Internetprogrammierung mit ASP und ASP.NET

PEARSON Education

ADDISON-WESLEY

Jürgen Bayer

Internetprogrammierung mit ASP und ASP.NET

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei
Der Deutschen Bibliothek erhältlich

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1

06 05 04 03 02

ISBN 3-8273-1826-2

ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany

Einbandgestaltung:	Vera Zimmermann, Mainz
Lektorat:	Christiane Auf, cauf@pearson.de
Korrektorat:	Simone Meißner, Großberghofen
Herstellung:	Philipp Burkart, pburkart@pearson.de
Satz und Layout:	mediaService, Siegen (www.media-service.tv)
Druck und Verarbeitung:	Nørhaven, Viborg (DK)

Printed in Denmark

Inhaltsverzeichnis

Teil II – Take that!

Teil III – Go ahead!

Vorwort

Die Programmierung von Internetanwendungen wird immer wichtiger. Viele Programme laufen heute schon im Internet und sind dort nicht mehr wegzudenken. Online-Shops wie der von Amazon sind gute Beispiele dafür.

Dieses Buch bietet »Nitty-Grittty«-Informationen zur Internetprogrammierung unter dem IIS (»Internet Information Server« bzw. »Internet Information Services«), dem Webserver von Microsoft. Ein wenig einschränkend muss ich zugeben, dass natürlich nicht alle Aspekte der Internetprogrammierung behandelt werden. Techniken bzw. Sprachen wie PHP, Perl, Java, Java Server Pages (JSP) etc., die nicht direkt in Verbindung mit dem IIS stehen, werden im Buch nicht behandelt. Das Buch konzentriert sich eben auf die Möglichkeiten, die Sie mit dem IIS haben. Aus der Sicht des Webservers ist das im Großen und Ganzen ASP (Active Server Pages) und ASP.NET (die neue ASP-Variante).

Da ASP alleine nicht ausreicht, um ansprechende und funktionierende Internetanwendungen zu entwickeln, behandelt das Buch auch die Möglichkeiten, die Sie auf der Clientseite besitzen. Neben den unverzichtbaren HTML-Grundlagen werden dabei auch Cascading Style Sheets (CSS), JavaScript und DHTML beschrieben. Da der Fokus des Buchs aber mehr auf der serverseitigen Programmierung liegt, werden komplexe Themen wie CSS und DHTML nur grundlegend besprochen.

Das Buch ist in drei Teile gegliedert: Der erste Teil zeigt die verschiedenen Wege zur Entwicklung von Internetprogrammen, beschreibt die Grundlagen des Internets (für Programmierer), die wichtigen Grundlagen von HTML und der clientseitigen Programmierung in HTML-Dokumenten.

Im zweiten Teil geht das Buch auf die klassische ASP-Programmierung ein, beschreibt also, wie Sie mit ASP-Skripten, die Sie in HTML-Code einbetten, dynamisch Webseiten erzeugen können. Ein wichtiges Thema in diesem Teil ist die Bearbeitung und Abfrage von Datenbanken mit den ActiveX Data Objects (ADO). Der zweite Teil konzentriert

sich auf das klassische ASP, das ohne weitere Vorbereitungen auf dem IIS ab Version 4 läuft. Ich gehe aber berreits in diesem Teil immer wieder auf die Programmierung unter .NET ein.

Im zweiten Teil wird damit das Grundlagenwissen vermittelt, das für den dritten Teil vorausgesetzt wird. Dieser beschreibt dann die Internetprogrammierung mit den neuen Features des .NET-Framework. Viele Programme, die unter dem klassischen ASP noch sehr aufwändig programmiert werden mussten, reduzieren sich unter ASP.NET auf wenige Zeilen Quellcode. Der dritte Teil des Buchs beschreibt die Grundlagen des .NET-Framework, erläutert die wichtigsten ASP.NET-Steuerelemente (die die meiste Arbeit selbst erledigen) und führt Sie in die Arbeit mit Datenbanken (eher: Datenquellen) und in die Datenbindung ein.

Sie werden sich u. U. fragen, warum die alte ASP-Welt in diesem Buch noch beschrieben wird. Die Antwort ist, dass ASP.NET auf ASP aufsetzt. Ein solides ASP-Wissen hilft beim Verständnis vieler ASP.NET-Features. ASP.NET-Programme können außerdem prinzipiell genau so aufgebaut werden wie klassische ASP-Programme. Einfache Webanwendungen werden vielleicht auch in Zukunft noch mit klassischem ASP entwickelt.

Für wen ist dieses Buch geschrieben?

Wie alle Bücher der Nitty-Gritty-Reihe ist dieses Buch für den Leser mit etwas Programmiererfahrung geschrieben. Es wird nicht vorausgesetzt, dass Sie ein absoluter Programmier-Profi sind. Die Grundlagen der Programmierung sollten Sie aber beherrschen. Lesen Sie u. U. erst mein (!) Buch »Programmierung« aus der Nitty-Gritty-Reihe ☺.

Die in den einzelnen Kapiteln verwendeten Programmiersprachen werden aufgrund ihrer Komplexität nur grundlegend erläutert. Wenn Sie diese Sprachen nicht kennen, lesen Sie einfach das entsprechende Buch aus der Nitty-Gritty-Reihe. Dort finden Sie z. B. Bücher zu C#, Visual Basic.NET, HTML und JavaScript.

So, nun viel Spaß mit der Internetprogrammierung.

Jürgen Bayer
juergen.bayer@addison-wesley.de

Teil I

Start up!

1 Einführung

Dieses Kapitel behandelt einige wichtige Grundbegriffe der Internetprogrammierung. Dazu gehört zunächst, dass Sie wissen, wo Sie weitere Informationen finden. Danach beschreibe ich, wie sich Internetanwendungen und klassische Anwendungen unterscheiden. Ich zeige dabei im Besonderen die Vorteile der Internetprogrammierung. Danach beschreibe ich die beiden grundsätzlichen Möglichkeiten der Internetprogrammierung: clientseitige und serverseitige Programmierung. Die im Buch im zweiten Teil behandelte Variante der serverseitigen Programmierung – ASP – erläutere ich schließlich noch grundlegend, was auch die Installation eines ASP-fähigen Webservers mit einschließt.

1.1 Einige wichtige Infos zuvor

Die Buchseite bei www.nitty-gritty.de

Über die Internet-Startseite von Nitty-Gritty (*www.nitty-gritty.de*) erreichen Sie über den Bücher-Link die Seiten zu den einzelnen Nitty-Gritty-Büchern. Auf der Seite zu dem Buch, das Sie gerade lesen, finden Sie zusätzliche Artikel (deren Themen nicht mehr ins Buch gepasst haben) und alle Beispiele, die ich für das Buch erarbeitet habe.

Die Buchbeispiele

Die Buchbeispiele finden Sie auf der Downloadseite dieses Buchs. Alle Beispiele sind in eine einzige Datei gepackt. Entpacken Sie diese Datei in einen Ordner Ihrer Wahl. Achten Sie aber darauf, dass Ihr Entpack-Programm die Ordnerstruktur des Archivs beibehält. Wenn Sie Winzip besitzen, können Sie dazu einfach die Zip-Datei mit der rechten Maustaste anklicken und den Befehl »Extrahiere in den Ordner xyz« wählen. Wenn Sie die Beispiele mit dem IIS ausprobieren wollen, legen Sie danach im IIS einen neuen virtuellen Webordner an, der auf diesen Dateiordner zeigt. Konfigurieren Sie diesen Webordner als Webanwendung. Wie das geht, beschreibe ich in Kapitel 3.

Die Version von ASP.NET

Der dritte Teil dieses Buchs behandelt die Programmierung mit ASP.NET. Zur Zeit der Drucklegung dieses Buchs lag mir die Betaversion 2.0, Build 9344 vor. Diese Version läuft bereits sehr stabil und enthält laut Microsoft alle Features, die das erste Release bieten wird. Viele Änderungen sind demnach nicht zu erwarten. Tatsächlich hatte ich während der Arbeit zu diesem Buch kaum Probleme mit ASP.NET. Trotzdem kann es natürlich sein, dass das eine oder andere Feature im ersten Release etwas anders programmiert wird, als ich es in diesem Buch beschreibe. Schauen Sie einfach auf der Buchseite bei *www.nitty-gritty.de* nach. Dort werde ich alle Unterschiede zwischen der verwendeten Betaversion und dem ersten Release für dieses Buch beschreiben, sobald das Release erscheint (was etwa Dezember 2001 der Fall sein wird).

Suchen im Internet

Viele Informationen zur Internetprogrammierung im Allgemeinen und zu JavaScript, CSS, DHTML und ASP im Besonderen finden Sie natürlich im Internet. Die mir bekannten wichtigen Webseiten gebe ich in den einzelnen Kapiteln natürlich an. In den Downloads zum Buch finden Sie auch eine HTML-Datei mit allen wichtigen Links.

Bei der Suche nach einer Problemlösung ist aber auch die Suche im Internet wichtig. Damit meine ich jetzt aber nicht die allgemeine Suche mit Altavista, Google oder einer anderen Suchmaschine. Auf speziellen Suchseiten können Sie wesentlich gezielter suchen.

Die Entwickler-Suchseite von Microsoft (*Search.Microsoft.com/Search/us/dev/default.asp*) ist dabei (für Microsoft-Themen) eine sehr wichtige Adresse. Hier können Sie im *MSDN* (Microsoft Developer Network) und in der *Knowledge Base* suchen. Das MSDN enthält alle Dokumentationen zu Microsoft-Produkten, ganze Bücher, Artikel und vieles mehr. Die Knowledge Base bietet sehr viele How-To-Artikel und Bug-Berichte. Hier finden Sie – auch bei komplexen Problemen – meist eine Lösung.

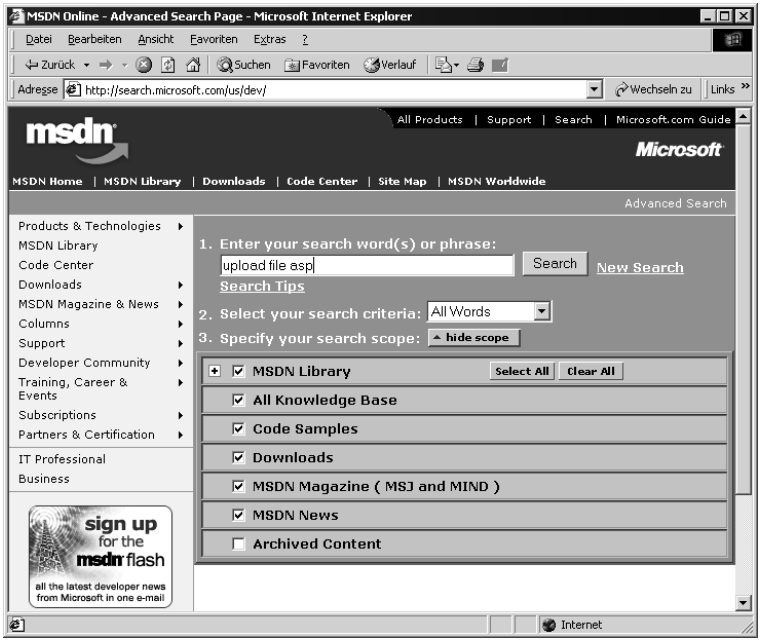


Bild 1.1: Suche auf der Entwickler-Suchseite von Microsoft

Abbildung 1.1 zeigt ein Beispiel für die Suche nach der Lösung für das Problem, Dateien in ASP zum Webserver upzuloaden. Achten Sie bei der Suche darauf, dass Sie im Suchbegriff einfach immer den Namen des Programmiersystems mit angeben (im Beispiel »asp«). Dann brauchen Sie sich nicht mit den komplexen Unteroptionen der einzelnen Suchbereiche auseinander zu setzen.

Wenn Sie bei Microsoft nicht fündig werden, können Sie noch in den Archiven von Newsgroups suchen. Die erweiterte Suche von »Google« (*groups.google.com/advanced_group_search*) ist dann eine der besten Adressen. Google hat übrigens das Archiv der bekannten und sehr guten Newsgroup-Suchmaschine »Deja« (*www.deja.com*) übernommen.

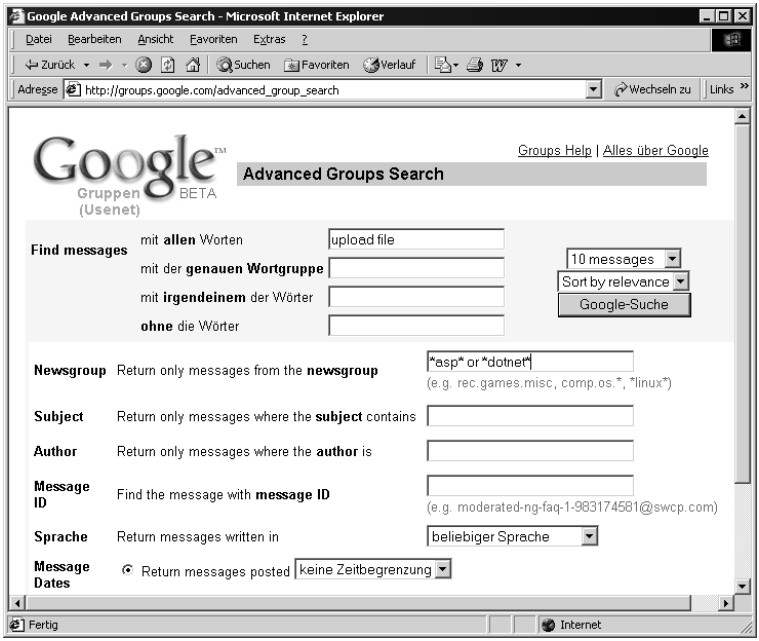


Bild 1.2: Recherche in der erweiterten Suche von Google

Die interessante Mischung aus deutschem und englischem Text liegt wohl daran, dass Google noch ein wenig an der Übersetzung arbeiten muss ...

In dem Feld NEWSGROUP können Sie Ihre Suche auf eine oder mehrere Newsgroups einschränken (mehrere trennen Sie einfach durch Kommata oder mit dem Schlüsselwort `Or`). Wenn Sie den Namen der Newsgroup nicht kennen oder in mehreren Gruppen suchen wollen, können Sie den Namen mit Wildcards maskieren, so wie es Abbildung 1.2 zeigt.

Im Suchbegriff selbst verwenden Sie den Namen des Programmiersystems besser nicht, weil Sie nicht davon ausgehen können, dass in allen Beiträgen dieser Name vorkommt. Wie Abbildung 1.2 zeigt, können Sie recht flexibel nach allen möglichen Kombinationen von Begriffen suchen. Das Beispiel sucht nach einer Lösung für das

Problem, in ASP Dateien upzuloaden. Für die Suche sollen im Beispiel Newsgroups verwendet werden, die »asp« oder »dotnet« im Namen haben. »dotnet« ist dabei ein wichtiger Begriff: Alle wichtigen .NET-Newsgroups führen diesen Begriff in ihrem Namen.

Newsgroup-ähnliche Foren für verschiedene Programmiersprachen (und andere Themen) finden Sie übrigens auf *www.Spotlight.de*. Auf diesen Foren ist (Zitat aus einem Forum) »ziemlich viel los«. Antworten auf Ihre Fragen können Sie hier recht schnell, meist innerhalb von wenigen Minuten, erwarten.

1.2 Clientseitige versus serverseitige Internetprogrammierung

Programme für das Internet können so entwickelt werden, dass diese auf dem Client (dem Anwenderrechner) oder dem Server laufen. Clientseitige Internetprogramme sind entweder echte (Windows-)Anwendungen, die über eines der Internetprotokolle (z. B. über TCP/IP, HTTP oder SOAP) mit anderen Anwendungen auf entfernten Rechnern kommunizieren, oder in HTML eingebundene Skripte. Serverseitige Programme werden normalerweise (wenigstens im Fall von ASP) in HTML-Seiten eingebunden, auf dem Webserver ausgeführt und im Webbrowser des Client angezeigt.

Clientseitige Programme

Programme, die *komplett* auf dem Computer des Anwenders ausgeführt werden, besitzen nicht allzu viel Bedeutung bei der Internetprogrammierung. Die Verwendung echter (Windows-)Anwendungen für solche Programme ist problematisch, da diese für jedes Betriebssystem separat entwickelt und vor der Benutzung auf dem Client installiert werden müssen. Wird die Anwendung umgearbeitet und verbessert, muss sie von allen Anwendern neu installiert werden. Der Administrationsaufwand für solche Programme ist also sehr hoch.

Dass clientseitige Internetanwendungen dennoch benötigt werden, zeigen Tauschbörsen wie Napster. Die Clients solcher Tauschbörsen sind meist normale Anwendungen, die über ein Internetprotokoll mit einem Server oder auch direkt mit anderen am Server angemeldeten Anwendungen kommunizieren.

Das Buch geht selbst nicht auf die Programmierung von echten Anwendungen ein, die über das Internet mit anderen Anwendungen kommunizieren. Auf der Website finden Sie aber entsprechende Artikel.

Clientseitige, in HTML eingebundene Programme werden aber häufig verwendet, um HTML-Seiten, die ohne Programmierung sehr statisch sind, dynamisch zu machen. Eine HTML-Seite mit einem bei einer Mausberührung aufklappenden Menü z. B. wird fast ausschließlich über eingebettete Script-Programme erzeugt (alternativ ist die Verwendung von Java Applets oder ActiveX-Steuerelementen). Sehr häufig werden Script-Programme in Verbindung mit ASP-Dokumenten verwendet, um einige Teile der Programmierung auf den Client auszulagern und damit die Ausführung der Anwendung zu beschleunigen.

Serverseitige Programme

Serverseitige Programme werden auf dem Webserver ausgeführt. Der Client ruft diese Programme normalerweise über einen Webbrowser auf. Das Ergebnis ist dann meist eine HTML-Seite, die im Webbrowser dargestellt wird. Die Artikelliste eines Online-Shops wird z. B. normalerweise auf dieser Art aufgebaut und dargestellt. Manche Programme geben aber auch Daten zurück, meist in Form von XML. Diese Daten kann der Client dann beliebig weiter verarbeiten.

HTML-Seiten werden meist von Programmen erzeugt, die CGI, Perl, PHP, ASP, JSP oder eine ähnliche Technologie bzw. Sprache verwenden. Dieses Buch geht ab dem nächsten Abschnitt grundlegend und ab Kapitel 7 intensiv auf ASP ein.

Eine neue Version serverseitiger Programme sind Webdienste. Webdienste sind im Prinzip einfache Funktionen, die – im Gegensatz zu normalen Funktionen – auf einem Webserver gespeichert sind und über das Internet aufgerufen werden. Das Ergebnis ist normalerweise ein XML-Dokument (siehe im nächsten Abschnitt) mit Daten, die vom Client weiter verarbeitet werden können. Webdienste werden ab Seite 23 grundlegend behandelt.

1.3 Was ist HTML, DHTML, CSS, XHTML, XML?

HTML (HyperText Markup Language) ist eine textbasierte Sprache zur Beschreibung des Inhalts und der Formatierung von Dokumenten. Im Prinzip können Sie ein HTML-Dokument mit einem Word-Dokument vergleichen: Ein solches Dokument enthält Text, Grafiken und Formatierungen. Der maßgebliche Unterschied ist, dass ein HTML-Dokument aus lesbarem Text besteht. HTML-Dokumente werden hauptsächlich im Internet eingesetzt und von Webbrowsern angezeigt. Das kennen Sie ja wahrscheinlich bereits. Die wichtigen Bestandteile des HTML-4.0-Standards werden in Kapitel 4 beschrieben.

DHTML (Dynamic HTML) ist keine Sprache wie HTML, sondern eine Erweiterung des HTML-Standards. Browser, die DHTML unterstützen, ermöglichen Script-Programmen (die in HTML eingebunden sind) den Zugriff auf alle Elemente des HTML-Dokuments über ein Objektmodell. HTML-Elemente werden dazu als Objekte betrachtet, deren Eigenschaften gelesen und geändert werden können. Mit JavaScript ist es über das DHTML-Modell z. B. kein Problem, die Hintergrundfarbe eines Verweises (Links) beim Herüberfahren mit der Maus zu ändern oder ein Menü beim Klicken mit der Maus aufzuklappen (wie Sie dies von einigen Websites her kennen). DHTML wird in Kapitel 6 beschrieben.

CSS (Cascading Style Sheets) ist eine Erweiterung von HTML, die es ermöglicht, den Stil (die Formatierung) von HTML-Elementen flexibler zu gestalten, als es mit HTML allein möglich ist. Über CSS-Stile können Sie einen Absatz z. B. ohne Probleme mit einer bestimmten Hintergrundfarbe und einem Rahmen versehen, was mit HTML alleine nicht möglich ist. CSS-Stile können zudem in externen Dateien verwaltet und in HTML-Dokumente gelinkt werden. Damit können Sie ein bestimmtes Layout für alle HTML-Dokumente einer Website festlegen und sehr einfach auch ändern. CSS wird in Kapitel 5 beschrieben.

Üblicherweise verwenden Websites eine Kombination aus HTML, CSS und DHTML:

- HTML erzeugt den Grundaufbau der Seite,
- Cascading-Style-Sheet-Dateien verwalten den Stil aller zur Website gehörenden HTML-Dateien (womit eine Änderung des Stils der Website recht einfach möglich ist),

- JavaScript-Programme reagieren auf bestimmte Ereignisse wie eine Mausbewegung auf einem HTML-Element,
- über DHTML erhalten diese Programme Zugriff auf alle Objekte der HTML-Seite und können diese dynamisch verändern (z. B. ein Menü dynamisch erzeugen und sichtbar machen).

XHTML (Extensible HyperText Markup Language) ist eine Weiterentwicklung von HTML 4.0. XHTML basiert auf HTML 4.0 und XML und verbindet die Vorzüge dieser beiden Sprachen. Da die aktuellen Webbrowser zurzeit und in der näheren Zukunft lediglich HTML 4.0 unterstützen, besitzt XHTML noch keine allzu große Bedeutung. XHTML wird deshalb in diesem Buch nicht beschrieben.

XML (Extensible Markup Language) ist eine Familie von Techniken zur Speicherung und Verarbeitung von strukturierten Daten in Textdateien. XML-Dateien sehen zwar ähnlich aus wie HTML-Dateien, besitzen aber eine grundlegend andere Bedeutung: HTML-Dateien enthalten unstrukturierte *Dokumente* (deren Bedeutung es ist, irgendwie dargestellt zu werden), XML-Dateien speichern strukturierte *Daten* (mit denen eine Anwendung prinzipiell alles machen kann, nicht nur darstellen). XML-Dateien können Sie sich vorstellen wie Datenbanken (die ja auch strukturierte Daten speichern). XML besitzt aber einige Vorteile, z. B. dass die Daten in Textform gespeichert werden und dass die Struktur der Daten absolut flexibel ist.

Jedes Programm, das XML-fähig ist, kann diese Daten auslesen und weiter verarbeiten. Das ist der große Vorteil von XML.

XML besteht nicht nur aus der Festlegung, wie Daten gespeichert werden sollen, sondern definiert auch andere Techniken. XSL-Stylesheets können eingesetzt werden, um das Layout von XML-Daten zu beschreiben. XML-Schemata werden verwendet, um die Struktur eines XML-Dokuments zu beschreiben (damit ein XML-fähiges Programm die Struktur erkennen kann).

XML ist für Programmierer wichtig, da viele Daten im Web mittlerweile in XML-Form versendet werden. Die neuen Webdienste, die Microsoft mit dem .NET-Framework ermöglicht hat, versenden angeforderte Daten z. B. in XML-Form.

XML ist so komplex, dass dieses Buch nicht darauf eingehen kann. Das XML-Buch der Nitty-Gritty-Reihe beschreibt XML sehr ausführlich.

1.4 Was ist ASP 2/3, was ist ASP.NET?

ASP 2/3 und ASP.NET sind Technologien für die Internetprogrammierung und keine Sprachen, wie einige Leser vielleicht vermuten. ASP erlaubt die Arbeit mit verschiedenen Programmiersprachen. ASP 2/3-Dokumente können Sie z. B. mit VBScript oder JScript programmieren. Für ASP.NET steht Ihnen eine große Zahl an Programmiersprachen, wie C#, Visual Basic.NET und Cobol.NET, zur Verfügung.

Mit ASP 2/3 und ASP.NET können Sie recht einfach und schnell serverseitige Internetprogramme wie z. B. einen Online-Shop entwickeln. ASP ist aber natürlich nicht nur auf das Internet beschränkt, sondern wird auch sehr häufig in Intranets verwendet. Viele Firmen setzen ASP-Programme für die Verwaltung und Abfrage wichtiger Firmendaten ein.

Der große Vorteil von ASP gegenüber normalen Windowsanwendungen ist, dass ASP-Programme in jedem zur verwendeten HTML-Version kompatiblen Webbrowser ausgeführt werden können. Ist der Webserver mit dem Internet verbunden, können ASP-Programme ohne weitere Vorkehrungen nicht nur im Intranet, sondern von jedem Platz dieser Welt verwendet werden. Voraussetzung dafür ist lediglich ein vorhandener Internetzugang. Die in den aktuellen Webservern integrierten Sicherheitsmechanismen, die z. B. eine Autorisierung mit Benutzername und Passwort erzwingen, ermöglichen einen abgesicherten Zugang zu den ASP-Programmen, so dass kein unbefugter Benutzer über das Internet auf die Firmendaten zugreifen kann.

Da ASP-Programme ohne Probleme mit aktuellen Microsoft-Technologien (ADO, COM, .NET) interagieren, sind der Programmierung eigentlich (bis auf eingeschränkte Darstellungsmöglichkeiten im Webbrowser) keine Grenzen gesetzt.

ASP-Programme werden immer auf dem Server ausgeführt und erzeugen normalerweise – das hängt von der Programmierung ab – reinen HTML-Code für den Client (den Webbrowser). Mit ASP ist es kein großes Problem, einen Online-Shop aufzubauen, der seine dynamischen Inhalte (wie z. B. eine Auflistung der Artikel) aus einer Datenbank ausliest (die natürlich in der direkten Nähe des Webservers verwaltet wird) und die Bestellungen der Kunden ebenfalls in einer Datenbank abspeichert. Alles, was Sie sich als normales Windowsprogramm vorstellen können, können Sie auch mit ASP bzw. ASP.NET für das Internet oder das Intranet programmieren. Sie sind lediglich ein wenig auf die Darstellungsmöglichkeiten in einem Webbrowser eingeschränkt. Aber das ist nur selten ein Problem. Die Möglichkeiten, die Dynamic HTML (DHTML) und die Cascading Style Sheets (CSS) bieten, heben die prinzipiellen Einschränkungen recht gut auf.

ASP 2 ist die ältere Variante, die Bestandteil verschiedener Microsoft-Webserver für ältere Betriebssysteme ist (z. B. NT 4). ASP 3 steht auf modernen Systemen wie Windows 2000 zur Verfügung.

ASP.NET ist eine Weiterentwicklung von ASP, die auf dem Microsoft .NET-Konzept basiert. Der grobe Unterschied zwischen ASP und ASP.NET ist, dass ASP-Programme direkt in HTML-Seiten integriert und vom Webserver interpretiert werden und ASP.NET-Programme echte, kompilierte Programme sind, die flexible und umfangreiche Steuerelemente nutzen. ASP.NET bringt eine wesentlich verbesserte Performance und eine wesentlich einfachere und flexiblere Programmierung mit sich. ASP.NET-Programme werden in einer echten Entwicklungsumgebung mit einer echten Programmiersprache entwickelt. Die Entwicklungsumgebung bietet sehr viele Features beim Programmieren und beim Debuggen. Eine große Anzahl spezieller Steuerelemente, deren Verwendung sehr einfach ist, erleichtert die Erstellung der Oberfläche. Diese Steuerelemente sorgen bei der Ausführung des Programms selbst dafür, dass für den Webbrowser auf dem Client passender HTML-Code erzeugt wird. Dynamische Webseiten, die auf Ereignisse wie einen Mausklick auf einen Schalter reagieren, sind mit ASP.NET nur noch ein sehr kleines Problem. Der Programmierer reagiert lediglich in Ereignisprozeduren auf die gewünschten Ereignisse, so wie es Visual Basic-, Delphi- oder .NET-Programmierer sowieso schon gewöhnt sind. Bei der Ausführung des

Programms wird je nach Browser sogar automatisch JavaScript-Code in das HTML-Dokument integriert. Damit wird das beim alten ASP oft notwendige »Hin und Her« (bezeichnet als »Roundtrip«) zwischen Client und Server auf ein Minimum reduziert.

1.5 Was sind Webdienste?

Webdienste sind Programme, die auf einem Webserver laufen und von beliebigen Anwendungen über das Internet verwendet werden können. Im Prinzip ist ein Webdienst eine einfache Funktion, die allerdings – anders als normale Funktionen – über das Internet von einem beliebigen (internetfähigen) Computer aufgerufen werden kann und die ihr Ergebnis ebenfalls über das Internet zurückliefert. Eine Bank könnte z. B. einen Webdienst anbieten, der den aktuellen Kurswert von Aktien liefert. Microsoft verfolgt damit eine Idee, die als »Software as a Service« bezeichnet wird: Eine Anwendung bedient sich verschiedener Webdienste um den Anwender mit Informationen und Features zu versorgen und muss selbst kaum noch Programmcode enthalten.

Der Aufruf des Webdienstes und das Zurücksenden der Daten erfolgen über SOAP, ein spezielles, normalerweise auf HTTP basierendes Protokoll, oder über das HTTP-Protokoll selbst. Deshalb können alle SOAP- bzw. HTTP-kompatiblen Anwendungen Webdienste verwenden. Microsoft bietet ein kostenloses »SOAP-Toolkit« an, mit dem die Programmierung von SOAP-fähigen Anwendungen recht einfach ist. Wenn Sie Ihre Anwendungen in einer .NET-Sprache entwickeln, können Sie stattdessen auf die im .NET-Framework enthaltenen SOAP-Klassen zurückgreifen um Webdienste zu verwenden. Natürlich können Sie mit dem .NET-Framework auch eigene Webdienste entwickeln. Das Buch behandelt die Programmierung von Webdiensten allerdings nicht.

1.6 ASP mit Komponenten

Die Nachteile von ASP 2/3-Programmen (komplexer Aufbau, langsame Ausführung, schlechte Debugging-Möglichkeiten etc.) können teilweise dadurch aufgehoben werden, dass diese Programme Kom-

ponenten verwenden, die in einer echten Programmiersprache entwickelt werden. Mit ASP 2 wird dazu meist das Microsoft COM-Konzept verwendet, neuere ASP-Programme verwenden .NET-Komponenten. COM-Komponenten können recht einfach mit Visual Basic 6 entwickelt werden, .NET-Komponenten mit einer beliebigen .NET-Programmiersprache.

Die Verwendung von COM-Komponenten wird in Kapitel 8 behandelt (die Erzeugung aus Aktualitätsgründen nicht mehr), die von .NET-Komponenten (die in .NET Assemblierungen heißen) beschreibt Kapitel 11. Die Programmierung solcher Komponenten behandelt das Buch allerdings nicht. Im C#- oder Visual Basic.NET-Buch der Nitty-Gritty-Reihe finden Sie dazu nähere Informationen. ASP.NET-Programme können neben Komponenten auch so genannte Code-Behind-Klassen verwenden, die das eigentliche Programm enthalten. Diese Klassen werden in Kapitel 11 beschrieben.

1.7 Voraussetzungen für ASP

Server

ASP-Programme werden immer auf einem Webserver ausgeführt. Die Installation eines ASP-fähigen Webservers auf dem Servercomputer ist also eine wichtige Voraussetzung. Die Ausführung von ASP 2/3-Programmen ist dann bereits möglich. Wenn Sie mit ASP.NET arbeiten wollen, ist auf dem Server zusätzlich die Installation des .NET-Framework notwendig.

Wollen Sie eine Entwicklungsumgebung nutzen, erfordert dies die Installation der »FrontPage-Servererweiterungen«, die der Entwicklungsumgebung die Möglichkeiten schaffen, mit dem Webserver auf die notwendige Art zu kommunizieren und die Programme zu debuggen.

Client

Auf dem Client werden nur eine Internetverbindung und ein Webbrowser benötigt. Wie aktuell der Webbrowser sein muss, hängt von der verwendeten HTML-Version ab. Wenn Sie in Ihren ASP-Dokumenten mit DHTML und Cascading Style Sheets arbeiten, benötigt der Client schon einen neueren Browser.

Entwicklungsumgebungen

ASP-Programme können Sie natürlich in einem einfachen Editor schreiben. Leider fehlen Ihnen dann alle Features, die eine moderne Entwicklungsumgebung bietet. Als Entwicklungsumgebung für ASP 2/3 können Sie Microsoft FrontPage verwenden. Damit besitzen Sie schon einen guten Editor für die Oberfläche und für den Programmcode. Die Debugging-Möglichkeiten sind allerdings sehr eingeschränkt. Interessanter für ASP 2/3 ist Microsoft Visual Interdev, das im Wesentlichen eine Erweiterung von FrontPage in Richtung ASP-Entwicklung ist. Visual Interdev übernimmt sehr viele Aufgaben für Sie, so dass Sie sich auf die eigentliche Programmierung konzentrieren können. Debuggen können Sie ASP-Anwendungen mit dem Microsoft Script Debugger, der normalerweise automatisch mit dem IIS installiert wird.

ASP.NET-Programme können Sie ebenfalls in einem einfachen Editor entwickeln. Diese Programme werden zwar immer kompiliert, diese Aufgabe übernimmt aber normalerweise ASP.NET automatisch wenn eine ASP.NET-Seite vom Browser zum ersten Mal angefordert wird. Wenn Sie in Ihren Programmen allerdings mit Komponenten arbeiten, müssen Sie diese über den Befehlszeilencompiler des .NET-Framework selbst kompilieren (CSC.EXE für C#-Programme, VBC.EXE für VB-Programme). Das Debuggen ist unter .NET auch sehr flexibel: der in das .NET-Framework integrierte Debugger bietet sehr viele Möglichkeiten zum Entwanzen von fehlerhaften Programmen. Eine echte Entwicklungsumgebung benötigen Sie also nicht unbedingt.

Alternativ können Sie zur Entwicklung von ASP.NET-Programmen auch Visual Studio.NET, die .NET-Entwicklungsumgebung von Microsoft, verwenden. Diese besitzt flexible Debugging-Features, hilft über das IntelliSense-Feature enorm beim Schreiben des Quellcodes und erleichtert vor allen Dingen die Gestaltung der Oberfläche. Wenn Sie also Visual Studio.NET besitzen (das wie Visual Interdev leider gekauft werden muss), sollten Sie es installieren.

Eine alternative Entwicklungsumgebung für ASP.NET (und auch für Windows-Anwendungen) finden Sie im Internet unter *www.icsharpcode.net/opensource/sd/*. Dieser kostenlose Editor beherrscht die wichtigsten Techniken und wird kontinuierlich weiterentwickelt.

Entwicklungs-Computer

Falls Sie in einer Firma mit einem bereits vorhandenen Intranet arbeiten, sollten Sie Ihren Entwicklungsrechner an das Intranet anschließen. Die Mitarbeiter Ihrer Firma können dann jederzeit Ihre Internetanwendungen testen. Lassen Sie sich dazu vom Systemadministrator eine IP-Adresse mit passender Subnetzmaske geben und stellen Sie diese in der Netzwerkkonfiguration für Ihre Netzwerkkarte ein.

Kapitel 2 geht noch ausführlich auf die wichtigen Internetgrundlagen ein.

Zum Anschluss an das Intranet muss Ihr Rechner mit einer Netzwerkkarte ausgerüstet sein und das TCP/IP-Protokoll installiert haben. Das TCP/IP-Protokoll können Sie, falls notwendig, über die Netzwerkeigenschaften nachinstallieren. Unter Windows 2000 finden Sie diese Eigenschaften, indem Sie im Start-Menü auf EINSTELLUNGEN, dann auf NETZWERK- UND DFÜ-VERBINDUNGEN, dann auf LAN-VERBINDUNG und schließlich auf den Schalter EIGENSCHAFTEN klicken. Unter Windows NT klicken Sie in der Systemsteuerung auf den Eintrag NETZWERK.

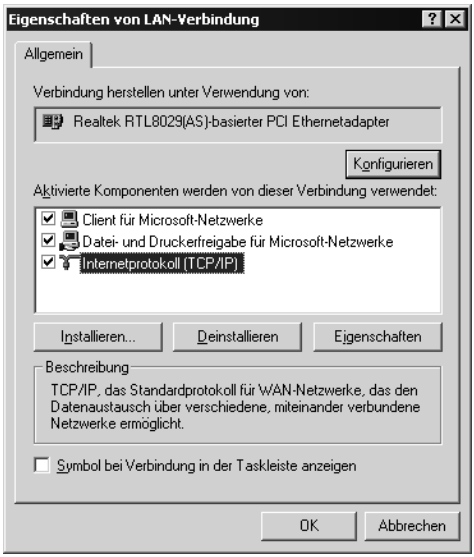


Bild 1.3: Die Netzwerkeigenschaften unter Windows 2000

Die IP-Adresse und die Subnetzmaske geben Sie in den Eigenschaften des TCP/IP-Protokolls ein. In einem Intranet müssen IP-Adressen eindeutig sein und Subnetzmasken zu den IP-Adressen passen. Achten Sie also darauf, dass Sie die Daten korrekt eingeben.

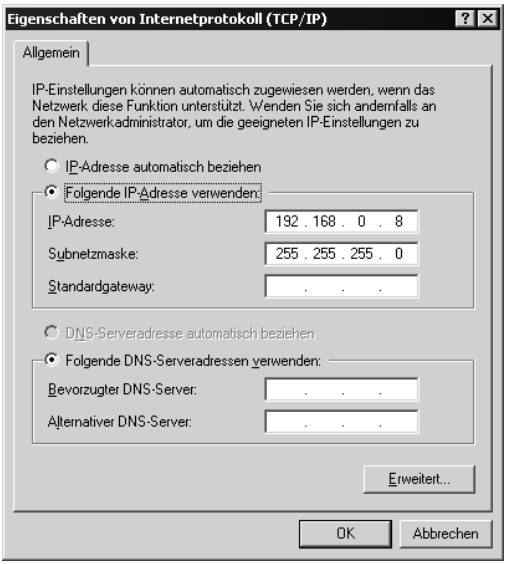


Bild 1.4: Eine typische TCP/IP-Einstellung für ein privates Intranet

Falls Sie nicht in einer Firma entwickeln, die bereits ein Intranet besitzt, können Sie sich mit mindestens zwei Rechnern ein eigenes Intranet aufbauen. Für die Entwicklung von Internetanwendungen reicht ein einziger Rechner zwar aus, ein zweiter Rechner erleichtert aber das Testen der Anwendung.

Wenn Sie nur ein kleines Netz aufbauen, das nicht mit anderen Netzen (außer dem Internet) verbunden ist, sollten Sie den für private Netze festgelegten IP-Bereich 192.168.0.0 bis 192.168.0.255 verwenden. Eine solche IP-Adresse ist eine so genannte Klasse-C-Adresse. Bei diesen Adressen adressieren die ersten drei Teile das Netzwerk und der letzte Teil den Rechner. Im letzten Teil (.000 bis .255) können Sie die Nummer frei vergeben. Als Subnetzmaske geben Sie dann 255.255.255.0 ein.

Testen Sie Ihr Intranet, indem Sie von einem Rechner aus auf der DOS-Eingabeaufforderung einen anderen Rechner »anpingen«. Geben Sie dazu einfach `ping` gefolgt von der IP-Adresse oder dem Namen des anderen Rechners ein. ping versucht dreimal, den Rechner anzusprechen, und liefert im Erfolgsfall eine Zeitangabe für die Rückmeldung des anderen Rechners (Abbildung 1.5).

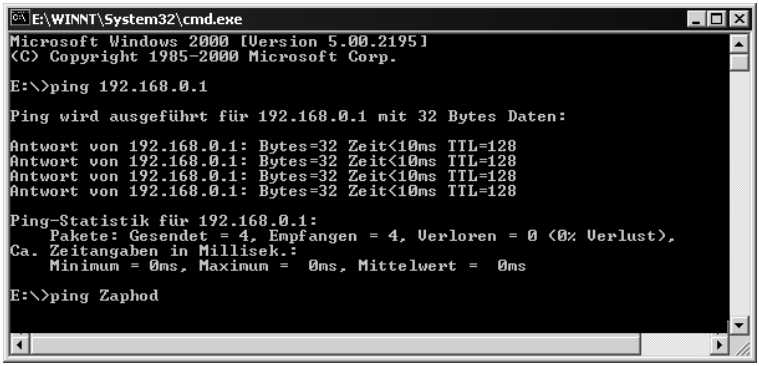


Bild 1.5: Ein erfolgreicher ping zu einem anderen Rechner im Intranet

Sie können den anderen Rechner auch mit dessen Namen ansprechen. Windows verwaltet die Namen der ans Netz angeschlossenen Rechner automatisch und wandelt diese bei Bedarf in die IP-Adresse um.

Falls ping Fehler meldet, ist entweder die angegebene IP-Adresse falsch oder Ihr Intranet funktioniert nicht. Überprüfen Sie dann alle IP-Adressen und die Subnetzmasken. Meist ist lediglich eine IP-Adresse nicht korrekt. Achten Sie besonders darauf, dass die Teile der Adresse, die zur Netzwerkadresse gehören (bei einem Klasse-C-Netz sind das die ersten drei Teile), auf allen Rechnern identisch sind.

1.8 Die Installation

1.8.1 Vorüberlegungen

Welche ASP-Version und welche Entwicklungsumgebung?

Vor der Installation sollten Sie zunächst überlegen, welche ASP-Version Sie installieren wollen bzw. müssen. Falls Sie das .NET-Framework

installieren und mit ASP.NET arbeiten wollen (was ich empfehle, da Sie dann beide ASP-Varianten verwenden können) und eine Entwicklungsumgebung installieren wollen, sollten Sie Visual Studio.NET den Vorzug geben. Mit dieser sehr umfangreichen Entwicklungsumgebung können Sie sehr einfach ASP 2/3- und ASP.NET-Programme entwickeln. Vor Visual Studio.NET muss dann unter Windows 95/98/Me/NT das Windows NT Option Pack, unter Windows 2000 müssen die Windows-Komponenten installiert werden. Auf allen Windows-Versionen ist danach noch die Installation des .NET-Framework notwendig.

Falls Sie das .NET-Framework nicht installieren, also nur mit ASP 2/3 arbeiten wollen, sollten Sie als Entwicklungsumgebung Visual Interdev 6 installieren (immer vorausgesetzt, Sie besitzen dieses Programm). Die erforderliche Installation des Windows NT Option Pack ist in Visual Interdev 6 bereits enthalten. Wollen Sie keine Entwicklungsumgebung verwenden, installieren Sie lediglich das Windows NT Option Pack. Debuggen können Sie auch ohne Visual Interdev mit Hilfe des im Option Pack enthaltenen Script Debuggers (der meist sogar noch besser funktioniert).

Wo installieren?

Für die Entwicklung von Internetanwendungen sollten Sie einen separaten Computer verwenden. Falls Sie in einer Firma arbeiten, sollten Sie auf keinen Fall den Server nutzen, den die Firma als ihren Webserver einsetzt. Beim Entwickeln von Internetanwendungen kommt es immer wieder vor, dass Sie den Webserver »abschießen« und den Rechner dann neu starten müssen. Außerdem müssen Sie für das Debuggen einige Einstellungen im Webserver so umstellen, dass der normale Betrieb sehr langsam wird. Entwickeln Sie also Ihre Internetanwendungen auf einem separaten Rechner, der mit einem eigenen Webserver und idealerweise auch mit einer Entwicklungsumgebung ausgestattet ist. Ist die Anwendung fertig gestellt, kopieren Sie diese einfach auf den Produktionsrechner.

Als Betriebssystem für den Produktionsserver sollten Sie Windows NT oder Windows 2000 verwenden. Die Sicherheitseinstellungen im IIS arbeiten eng mit den Sicherheitseinstellungen des Betriebssystems zusammen. Unter Windows NT und Windows 2000 können Sie – im Gegensatz zu den anderen Windows-Versionen – das NTFS-Datei-

system verwenden und somit für jede Datei und jeden Ordner separate Zugriffsrechte einstellen. Auf Produktionsservern, die von vielen verschiedenen Anwendern verwendet werden, ist dies ein absolutes Muss.

Für den Entwicklungsrechner ist die Verwendung von NTFS zwar nicht unbedingt notwendig. Ohne NTFS können Sie aber die Sicherheits-Features nicht so nutzen, wie es auf dem Produktionsrechner der Fall ist. Ich rate deshalb, auch den Entwicklungsrechner mit Windows NT oder Windows 2000 und mit NTFS auszustatten.

Eine Entwicklungsumgebung können Sie auch auf einem anderen Rechner installieren als auf dem Webserver. Das macht u. U. dann Sinn, wenn mehrere Entwickler an demselben Projekt arbeiten. Das Entwickeln ist dann relativ problemlos. Probleme bereitet das Debuggen, da der Debugging-Prozess auf einem anderen Rechner ausgeführt werden muss. Falls Sie nicht im Team arbeiten, rate ich Ihnen davon also ab.

1.8.2 ASP 2 unter Windows 95, 98, Me, NT

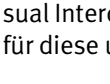
ASP 2 ist Bestandteil des Internet Information Server (IIS) 4.0 (Windows NT) und des Personal Web Server (PWS) 4.0 (Windows 95). Diese kostenfreien Webserver finden Sie auf neueren (NT-)Installations-CDs, in neueren Service Packs oder – wahrscheinlich eher – im »Windows NT Option Pack«. Dieses Paket steht – unabhängig vom Namen – für Windows 95, NT 4 Workstation und NT 4 Server zur Verfügung. Für Windows Me beschreibt Microsoft (unter *support.microsoft.com/support/kb/articles/Q266/4/56.asp*), dass das Option Pack mit einigen Einschränkungen installiert werden kann. Für Windows 98 wird Ähnliches gelten.

Sie können das Option Pack separat oder, falls Sie im Besitz von Visual Interdev sind, über das Visual-Interdev-Setup installieren.

Visual Interdev

Falls Sie Visual Interdev 6 besitzen, sollten Sie diese Entwicklungsumgebung auch verwenden, wenn Sie nicht mit ASP.NET arbeiten wollen. Visual Interdev nimmt Ihnen viele Aufgaben ab und erleichtert die Gestaltung der Oberfläche und das Schreiben von Programm-

code. Die Installation ist allerdings sehr empfindlich und stark von der Konfiguration Ihres Rechners abhängig. Falls Sie nicht korrekt installieren, funktionieren häufig einige wichtige Features wie das Debuggen nicht. Microsoft beschreibt im Artikel »Visual InterDev 6.0 Installation Tips«, den Sie im Internet unter der Adresse *support.microsoft.com/support/kb/articles/Q243/8/98.asp* finden, wie Sie Visual Interdev korrekt installieren. In diesem Buch ist leider kein Platz für diese umfangreiche Beschreibung.



Achten Sie darauf, dass Sie genau nach dieser Installationsanleitung arbeiten. Installieren Sie idealerweise auf einer frischen Windows-Version. Installieren Sie nur die von Microsoft in der Installationsanleitung empfohlenen FrontPage- Servererweiterungen. Nur dann wird das Debuggen funktionieren. Installieren Sie nach der erfolgreichen Visual Interdev-Installation möglichst kein Update der FrontPage-Servererweiterungen. In meinem Fall war der Webserver nach der Installation eines Updates leider nicht mehr zu gebrauchen und lieferte nur noch Fehlermeldungen. Webserver sind eben sehr empfindlich ...

Das Windows NT Option Pack

Falls Sie Visual Interdev nicht verwenden wollen, können Sie das Windows NT Option Pack auch separat installieren. Sie finden das Option Pack im Internet unter der Adresse *www.microsoft.com/msdownload/ntoptionpack/askwiz.asp*. Achten Sie darauf, dass Sie die für Ihr System korrekte Variante herunterladen. Vor der Installation des Option Pack müssen Sie den Internet Explorer ab Version 4.01 und unter NT das Service Pack ab Version 3 installieren.

Starten Sie die Installation über die Datei SETUP.EXE. Falls Sie unter Windows NT installieren und ein Service Pack ab Version 4 installiert haben, meldet das Setup, dass das Produkt noch nicht mit Service Pack 4 getestet wurde (bei neueren Versionen des Option Pack fällt diese Meldung u. U. weg). Sie können trotzdem ohne Probleme installieren. Nachdem Sie dem Lizenzvertrag zugestimmt haben, wählen Sie auf jeden Fall die benutzerdefinierte Installation. Sie sollten hier die FrontPage-Servererweiterungen zunächst abwählen, damit

Sie später die aktuelle Version installieren können. Dieses Vorgehen wird von Microsoft für die Installation empfohlen (zumindest für den sehr empfindlichen Site Server, der auf dem IIS aufsetzt, siehe unter *support.microsoft.com/support/siteserver/install/install_ss3.asp*). Ändern Sie die Voreinstellung der zu installierenden Komponenten wie folgt:

- Wählen Sie die FrontPage-Servererweiterungen ab;
- wenn Sie unter Windows NT Workstation oder Windows 95/98/Me installieren, sollten Sie auf jeden Fall unter dem Eintrag »Personal Web Server (PWS)« in den Teilkomponenten den Eintrag »Internetdienst-Manager« wählen. Falls Sie dies versäumen, besitzen Sie später nur sehr eingeschränkte Administrations-Features, die für die ASP-Programmierung nicht ausreichen.

Installieren Sie nach dem Option Pack die aktuellen FrontPage-Servererweiterungen, die Sie unter der Adresse *www.microsoft.com/frontpage/fpse* finden.

Nach der Installation können Sie den IIS testen, indem Sie in einem Webbrowser einfach als Adresse »http://localhost« eingeben. Wenn der Webbrowser eine Willkommensseite anzeigt, ist alles in Ordnung.

1.8.3 ASP 3 unter Windows 2000

ASP 3 ist Teil des IIS 5, der mit Windows 2000 mitgeliefert wird. Sie finden den IIS in den Windows-Komponenten. Öffnen Sie die Systemsteuerungen, wählen Sie den Eintrag SOFTWARE und klicken Sie dann auf WINDOWS-KOMPONENTEN HINZUFÜGEN/ENTFERNEN. Installieren Sie dort die Internet Informationsdienste, idealerweise mit allen Unteroptionen.

Wie beim IIS 4 können Sie den IIS 5 testen, indem Sie in einem Webbrowser einfach als Adresse »http://localhost« eingeben. Zeigt der Webbrowser eine Willkommensseite an, ist alles in Ordnung.

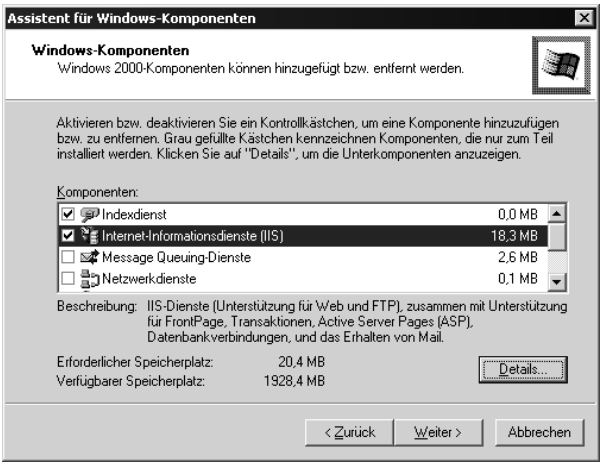


Bild 1.6: Der Assistent für die Installation der Windows-Komponenten

1.8.4 ASP.NET

Für ASP.NET ist die Installation des .NET-Framework notwendig. Zuvor müssen Sie den IIS installieren, wie ich es bereits im vorigen Abschnitt beschrieben habe. Das aktuelle .NET-Framework finden Sie unter der Adresse *msdn.microsoft.com/net*. Klicken Sie hier auf den Link ».NET Framework SDK« und wählen Sie dann das ».NET Framework Full SDK[1]«, das auch einige für Entwickler wichtige Tools enthält. Das Setup ist mehr als 100 MB groß, gehen Sie während des Download also irgendwo einen Kaffee trinken. Falls Sie im Besitz von Visual Studio.NET sind, müssen Sie das .NET-Framework nicht separat installieren. Diese Arbeit nimmt Ihnen die Visual Studio-Installation ab. In neuen Windows-Versionen wie XP wird das .NET-Framework übrigens Bestandteil des Betriebssystems sein. Das SDK müssen Sie dann allerdings wahrscheinlich trotzdem herunterladen und installieren.

Starten Sie zur Installation einfach die Datei SETUP.EXE. Die Installation ist recht einfach, für die wenigen Optionen können Sie die Voreinstellungen übernehmen. Das .NET-Framework-SDK benötigt ca. 200 MB auf der Festplatte.

1. SDK = Software Development Kit

Visual Studio.NET

Vor der Installation von Visual Studio.NET müssen Sie den IIS installieren (Seite 32). Legen Sie dann einfach die erste CD des Installationspakets ein und rufen Sie SETUP.EXE auf, falls die Installation nicht automatisch startet. Das Installationsprogramm fordert Sie dann auf, die CD mit dem Update der Windows-Komponenten einzulegen, die Bestandteil des Pakets ist. Nach dem Update dieser Komponenten werden Sie erneut aufgefordert, die erste Installations-CD einzulegen. Die Installation läuft problemlos ab, benötigt aber gut 1 GB freien Platz auf der Festplatte. Zeit sollten Sie sich allerdings nehmen: Die Installation benötigt normalerweise mehrere Stunden.

Installieren Sie auf jeden Fall vor dem .NET-Framework bzw. vor Visual Studio.NET den IIS. Eine nachträgliche Installation des IIS bringt viel Ärger und einen großen Zeitaufwand mit sich, da das Setup-Programm einige für die Ausführung von ASP.NET-Seiten erforderlichen Komponenten bei fehlendem IIS nicht installiert.

2 Internet-Grundlagen

Zur Internetprogrammierung gehört auch das Wissen um die wichtigsten Grundlagen im Internet. Derjenige, der z. B. das HTTP-Protokoll einigermaßen versteht, kann viel besser mit der Kommunikation zwischen Client und Server umgehen (was im Besonderen das Verständnis der HTTP-Fehler mit einschließt).

Ich gehe in diesem Kapitel davon aus, dass Sie wissen, was das Internet und was ein Intranet ist. Ich beschreibe deshalb nur die wichtigsten Protokolle und zeige, wie Sie den IIS administrieren.

2.1 RFCs

Alles, was irgendwie mit dem Internet zusammenhängt, wird in technischen Dokumenten beschrieben, die als »RFC« (Request For Comment) bezeichnet werden. Neben Diskussionen über neue Forschungsprojekte, Berichten über den Zustand des Internet und anderem werden dort auch die Internet-Protokolle sehr ausführlich dokumentiert. Ich gebe auf den folgenden Seiten immer wieder ein RFC-Dokument an, in dem Sie weitere Informationen finden (falls Ihnen die Informationen in diesem Buch nicht ausreichen, was ja eigentlich gar nicht sein kann ...).

RFC-Dokumente werden einfach nummeriert. RFC 0791 beschreibt z. B. das IP-Protokoll. Beim Verleger der RFCs, dem RFC-Editor, finden Sie eine Möglichkeit, die einzelnen RFCs einzusehen. Gehen Sie dazu zur Seite *www.rfc-editor.org/overview.html*. Klicken Sie auf den SEARCH-Link und geben Sie auf der Suchseite einen Suchbegriff wie z. B. die RFC-Nummer ein. Im weiteren Verlauf nehme ich immer wieder Bezug auf RFC-Dokumente.

Beachten Sie, dass RFCs kontinuierlich weiterentwickelt werden. Finden Sie im rechten Bereich des Suchergebnisses einen Eintrag »Obsoleted by«, ist der betreffende RFC mittlerweile durch einen neuen ersetzt worden.

Im RFC 2800 (zum Zeitpunkt der Erstellung dieses Buchs aktuell) finden Sie übrigens eine Übersicht über die RFCs, die die Internet-Protokolle betreffen.

2.2 Die IP-Adresse

Um jeden Rechner im Internet eindeutig adressieren zu können, besitzt dieser eine IP-Adresse. In einem TCP/IP-Netz wird jeder Computer über eine eindeutige IP-Adresse identifiziert. Diese Adresse ist ein 32-Bit-Dezimalwert, der häufig in Form von vier durch Punkte getrennte Bytewerte dargestellt wird. Eine typische IP-Adresse ist z. B. 128.66.12.1. IP-Adressen werden in Klassen unterteilt, die am ersten Byte erkannt werden:

- Ein Wert kleiner als 128 im ersten Byte bezeichnet eine Class-A-Adresse. Bei einer solchen Adresse adressiert das erste Byte das Netzwerk, die restlichen drei Bytes bezeichnen einen Rechner im Netz. Damit existieren maximal 128 Class-A-Adressen, die jedoch jede für sich Millionen von angeschlossenen Rechnern adressieren können.
- Ein Wert von 128 bis 191 bezeichnet eine Class-B-Adresse, bei der die ersten beiden Bytes das Netzwerk adressieren. Es gibt Tausende von Adressen der Klasse B und jede dieser Adressen kann maximal 65.536 angeschlossene Rechner adressieren.
- Eine Class-C-Adresse wird durch einen Wert von 192 bis 223 im ersten Byte gekennzeichnet. Class-C-Adressen nutzen die ersten drei Bytes als Netzwerknummer, wonach Millionen Class-C-Adressen existieren, die jede für sich jedoch nur maximal 255 Rechner adressieren können.
- Adressen, die einen Wert größer als 223 im ersten Byte enthalten, sind für besondere Zwecke reserviert. Diese Multicast-Adressen können z. B. Gruppen von Computern gleichzeitig ansprechen.
- Eine besondere Class-C-Adresse ist 192.168.0.x. Diese Adresse ist für private Zwecke (z. B. in einfachen Intranets) reserviert und wird im Internet nicht verwendet.

In einem Intranet ohne direkten Zugang zum Internet können Sie die IP-Adressen ohne weiteres frei vergeben. Sie müssen lediglich darauf achten, dass der Typ der Adresse (Class A, B oder C) und die Subnetzmaske[1] gleich sind. Ist einer der Rechner indirekt (z. B. über eine ISDN-Karte oder ein TDSL-Modem) an das Internet angeschlossen, sollten Sie den Adressbereich für private Adressen (192.168.0.0 bis 192.168.0.255) verwenden, um keine Probleme zu verursachen.

Für Rechner, die direkt an das Internet angeschlossen sind, muss eine IP-Adresse vom DE-NIC, dem deutschen Registrationsservice für IP-Adressen (*www.nic.de*) oder vom Internet-Provider bezogen werden, da diese weltweit eindeutig sein muss. Rechner, die über ein Gateway oder einen Provider an das Internet angeschlossen sind, können ihre IP-Adresse auch dynamisch über einen DHCP-Server beziehen.

IP-Namensauflösung

Wenn ein Client eine Hostadresse in einer IP-Anforderung verwendet, muss das System dafür sorgen, dass diese Adresse in die zugehörige IP-Adresse umgewandelt wird. In lokalen Netzen funktioniert die Namensauflösung meist schon automatisch oder über einen Nameserver. Im Internet werden dazu DNS-Server verwendet.

2.3 IP-Ports

Um verschiedene Dienste auf einem Server ansprechen zu können und um Daten zum richtigen Programm auf dem Client zurücksenden zu können, werden so genannte Ports verwendet. Ein Port ist einfach eine Dezimalzahl, die einen Dienst oder ein Programm identifiziert. Es gibt reservierte, so genannte »Well Known Ports«, mit einer Nummer unterhalb von 256. Diese Ports adressieren bekannte Dienste wie WWW, SMTP und FTP. Der WWW-Dienst eines Webservers wird z. B. immer über den Port 80 adressiert, der FTP-Dienst verwendet den Port 21. Wenn Sie in einem Webbrowser eine Webadresse wie

1. Subnetzmasken werden für Subnetze verwendet, über die die Administration von IP-Adressen erleichtert wird, da diese blockweise an die Subnetze vergeben und dort selbst verwaltet werden.

www.nitty-gritty.de eingeben, macht Ihr Webbrowser daraus automatisch die Adresse *www.nitty-gritty.de:80*. Der Doppelpunkt trennt die IP-Adresse vom Port. Das Betriebssystem auf dem Server erkennt an der 80, dass der WWW-Dienst angesprochen wurde, und übergibt die Daten an diesen Dienst.

Ports mit einer Nummer größer als 255 können beliebig verwendet werden[1]. Programme und Dienste, die nicht zum Internet-Standard gehören, wie z. B. der Microsoft SQL Server, nutzen eine solche freie Portnummer. Ein Webbrowser reserviert sich selbst einen freien Port, dessen Nummer – wie Sie beim UDP- und beim TCP-Protokoll ab Seite 41 noch sehen – mit zum Server übertragen wird. Der Server sendet die Ergebnisdaten dann genau an diesen Port. Deshalb ist es möglich, mit mehreren Instanzen eines Webbrowsers gleichzeitig zu arbeiten, ohne dass diese durcheinander geraten.

2.4 Die (für Programmierer) wichtigsten Internetdienste

Im Internet läuft eine Vielzahl an Servern, die die verschiedensten Dienste anbieten. Die wichtigsten davon werden hier kurz beschrieben.

2.4.1 Das World Wide Web

Unzählige Webserver tragen das World Wide Web (WWW) im Internet. Diese Server verwalten HTML-Seiten, multimediale Daten und Internetprogramme. Ein Webserver ist ein relativ einfaches Programm. Er horcht auf dem Port 80 auf eingehende HTTP-Nachrichten und wertet diese aus. Ein Client sendet über das HTTP-Protokoll (RFC 2616, siehe ab Seite 44), das auf TCP/IP aufsetzt, eine Anforderung an den Server. Darin wird der Server aufgefordert, entweder eine bestimmte Datei zum Client zu senden, gesendete Daten als Datei abzuspeichern oder ein Programm auszuführen. Multimediale Inhalte, die oft mit HTML-Seiten verknüpft sind, werden wie die HTML-Seiten selbst einfach zum Client gesendet. Der Webbrowser auf dem Client ist dafür verantwortlich, die empfangenen Daten korrekt zu interpretieren und darzustellen.

1. Früher waren noch die Ports 256 bis 1024 für UNIX-typische Dienste reserviert, was aber heute nicht mehr gilt.

Die meisten Webserver können Programme ausführen, die auf dem Server gespeichert sind. Die unterschiedlichen Server unterstützen dazu verschiedene Technologien, wie CGI, Perl, PHP, JSP und ASP. Ein serverseitiges Programm erzeugt bei der Ausführung meist gleich noch HTML-Code als Ergebnis und sendet diesen an den Client zurück. Der Webserver ermöglicht damit u. a. die Erstellung dynamischer Webseiten, die – im Gegensatz zu statischen Webseiten – immer aktuelle Informationen unterhalten. Mit diesem Thema beschäftigt sich das Buch noch sehr ausführlich.

Der Server ist mit verschiedenen Ordnern verknüpft, die die Webdateien enthalten. Wenn der Server läuft, kann ein Client diese Dateien über die IP-Adresse oder den Namen des Servers (sofern die IP-Namensauflösung funktioniert) abrufen. Webdateien werden ausgehend von einem logischen Root-Ordner in virtuellen Unterordnern verwaltet. Der Unterordner muss deshalb in vielen Fällen im Webbrowser angegeben werden. Eine Datei DEFAULT.HMT, die direkt im Root-Ordner des Servers TRILLIAN (im Intranet) gespeichert ist, wird im Webbrowser so abgerufen:

```
http://Trillian/default.htm
```

Eine Datei, die in einem virtuellen Unterordner mit Namen SHOP gespeichert ist, wird so abgerufen:

```
http://Trillian/Shop/default.htm
```

Wenn die Namensauflösung nicht funktioniert (z. B. weil einem Rechner im Internet noch kein Domänenname zugeordnet wurde), können Sie alternativ auch die IP-Adresse verwenden:

```
http://192.168.0.1/Shop/default.htm
```

Jeder Rechner im Internet kann einen Webserver verwalten. Dazu ist lediglich die Installation des Webservers selbst (z. B. des IIS) und eine Internetanbindung notwendig. Wenn Sie auf einem an das Internet angeschlossenen Rechner einen Webserver betreiben, kann jeder Internetnutzer die dort verwalteten HTML-Dateien und Programme über die IP-Adresse Ihres Rechners abrufen. Falls Sie den IIS auf einem Rechner installiert haben, der eine Internetverbindung besitzt, können Sie dies einfach einmal ausprobieren. Öffnen Sie die Internetverbindung, lesen Sie die meist dynamisch vergebene IP-

Adresse auf dem DOS-Prompt über das Programm IPCONFIG aus, besuchen Sie einen Freund oder eine Freundin mit Internetanschluss und geben Sie in deren Webbrowser Ihre IP-Adresse ein. Sie sehen dann die Startseite Ihres Webservers.

2.4.2 FTP

Unzählige FTP[1]-Server bilden den FTP-Dienst des Internet. Ein FTP-Server ermöglicht einfach, dass ein entfernter Rechner über eine geeignete Clientanwendung freigegebene Dateien downloaden und eventuell auch Dateien in bestimmte Ordner uploaden kann. Da der Dateidownload mittlerweile immer mehr über HTML-Dokumente und Webbrowser realisiert wird, nimmt die Bedeutung von FTP zunehmend ab. Das FTP-Protokoll wird im RFC 959 beschrieben.

2.4.3 Mail

Maildienste werden hauptsächlich von SMTP-, POP- und IMAP-Servern angeboten. Ein im Netzwerk des Netzbetreibers laufender SMPT-Server übernimmt meist das Weitersenden von E-Mails, nachdem ein Mailclient die Mail zum SMPT-Server übertragen hat. Der Server verwendet dabei das Simple Mail Transfer Protocol (SMTP), das auf TCP/IP aufsetzt und ein simples textbasiertes Protokoll ist. Dieses Protokoll ist im RFC 821 definiert. Auf der Seite 52 beschreibe ich die Grundlagen von SMTP.

In einem Netzwerk eingegangene Mails werden auf einem POP-Server gespeichert. Dieser Server ermöglicht einem Client, die Mails über das POP-Protokoll abzufragen. Ein Problem von POP-Servern ist allerdings, dass ein Mailclient die Mails herunterlädt und auf dem lokalen Rechner speichert. Will ein Anwender seine gespeicherten Mails von einem anderen Rechner aus abrufen, ist dies mit einem POP-Server prinzipiell nicht möglich. Dieses Problem löst das IMAP-Protokoll. Eingegangene Mails bleiben auf einem IMAP-Server gespeichert und können deshalb auch von beliebigen Rechnern aus abgerufen werden.

1. File Transfer Protocol

2.5 Einfache Internetprotokolle

Das Internet basiert auf der untersten Ebene auf dem IP-Protokoll. Alle Daten werden über dieses Protokoll verwendet. Auf IP setzen die Protokolle UDP und TCP auf, die IP um wichtige Features erweitern. Höhere Protokolle, wie HTTP und SMTP, basieren übrigens wieder auf UDP oder TCP. Die folgenden Abschnitte beschreiben die niedrigen Protokolle, die höheren werden ab Seite 44 beschrieben.

2.5.1 Das IP-Protokoll

Damit die Kommunikation zwischen den verschiedensten Rechnern im Internet überhaupt funktioniert, werden Daten im Internet über das standardisierte IP[1]-Protokoll versendet. Bei diesem Protokoll werden die zu sendenden Daten in einem Paket (einem Datagramm) verpackt und mit zusätzlichen Informationen wie der Adresse des sendenden und der Adresse des Zielrechners versehen. Anhand dieser Informationen können Router die einzelnen Datagramme durchs Internet leiten. Das IP-Protokoll besitzt keine Möglichkeit, die Zustellung der Daten zu garantieren, empfangene Daten auf Fehler zu überprüfen und Prozesse auf dem Zielrechner zu adressieren. Die maximale Länge der Daten ist zudem beschränkt. Das IP-Protokoll arbeitet auf der untersten Ebene. Programme können dieses Protokoll nicht direkt nutzen. Dafür stellt das Betriebssystem die höheren Protokolle UDP und TCP zur Verfügung.

Das IP-Protokoll wird im RFC 791 beschrieben.

2.5.2 Das UDP-Protokoll

Das UDP-Protokoll (User Datagram Protocol) setzt auf dem IP-Protokoll auf und erweitert dieses um Informationen über den Ursprungs- und den Zielport, um eine Header-Prüfsumme und um eine Längenangabe. Mit UDP kann ein Programm lediglich Daten an einen bestimmten Port senden und auf einem Port empfangen. Eine Überprüfung der Daten auf das korrekte Versenden ist nicht möglich. Zudem ist die Größe der Daten auf die Maximalgröße der Datagramme beschränkt. UDP sendet die Daten ohne zu überprüfen, ob

1. Internet Protocol

der Empfänger überhaupt zum Empfang bereit ist. Ist der Empfänger nicht bereit, gehen die gesendeten Daten einfach verloren.

UDP wird für eine performante Übertragung geringer Datenmengen verwendet. Um eine relative Übertragungssicherheit zu ermöglichen, geben Serverdienste, die UDP verwenden, möglichst immer eine Antwort auf eine Anforderung. Geht eine Antwort beim Client ein, wird das als erfolgreiche Übertragung gewertet. Geht keine Antwort ein, wird die Anforderung einfach erneut gesendet. Das bei TCP wesentlich aufwändigere Verpacken der Daten und der Verbindungsaufbau vor dem Senden erfordern oft mehr Verwaltungsaufwand als das eventuell erneute Senden bei UDP.

Eine Beschreibung des UDP-Protokolls finden Sie im RFC 768.

2.5.3 Das TCP-Protokoll

Das wesentlich häufiger verwendete TCP-Protokoll (Transmission Control Protocol) setzt wie UDP ebenfalls auf IP auf. TCP erweitert IP um:

- die Adressierung von Ports,
- die garantierte, fehlerfreie Zustellung
- und die Fähigkeit, große Datenmengen in mehrere Pakete aufgeteilt zu versenden.

Die zu sendenden Daten werden in kleine Pakete aufgeteilt. Jedes Paket erhält eine Sequenznummer. Der Empfänger kann die einkommenden Pakete an Hand dieser Nummer in der richtigen Reihenfolge zusammensetzen. Da jedes Paket zusätzlich mit einer Prüfsumme versehen ist, kann der Empfänger überprüfen, ob die Daten auf ihrem Weg eventuell beschädigt wurden. Erhält der Empfänger beschädigte Pakete oder sind Pakete verloren gegangen, fordert er die fehlenden Pakete einfach nach einer gewissen Zeit vom Sender nach. Damit ist die Zustellung von Daten bei TCP sehr sicher.

Um das Versenden der Daten von Rechner A zu Rechner B zu garantieren, baut TCP zunächst eine Verbindung zum Zielrechner auf. Dazu sendet TCP zunächst ein Segment an Rechner B mit der Aufforderung, die Sequenznummern der folgenden Pakete zu synchronisieren. Der Zielrechner erkennt dies als Verbindungswunsch, liest die Startse-

quenznummer[1] des Sender aus und antwortet mit einem Segment, in dem er Rechner A die Startsequenznummer der Pakete seiner potentiellen Antwort mitteilt. Daran erkennt Rechner A die Bereitschaft und beginnt mit dem Senden der Daten. Die einzelnen Pakete können dabei in loser Reihenfolge versendet werden und gehen häufig unterschiedliche Wege durchs Internet. Die Reihenfolge, in der die Pakete ankommen, ist also nicht festgelegt. An Hand der Sequenznummern kann der Zielrechner die Daten korrekt zusammensetzen.

Nach dem Senden der Daten sendet TCP ein Segment mit der Information, dass keine Daten mehr folgen. Der Zielrechner sendet daraufhin das Ergebnis der Anforderung (vielleicht eine HTML-Seite), natürlich wieder in Pakete unterteilt. Danach wird die Verbindung mit dem Senden eines Segments abgeschlossen, das wiederum Rechner A mitteilt, dass keine Daten mehr folgen.

Dieser Handshake zwischen den beteiligten Rechnern sichert ab, dass die Daten auch beim jeweiligen Zielrechner ankommen. Wie bei UDP wird der Dienst des Zielrechners und die Anwendung auf dem Client über Ports adressiert, die dem TCP-Paketen beigefügt werden.

Das TCP-Protokoll wird im RFC 793 beschrieben.

2.6 Socket-Dienste

Auf den einzelnen Rechnern sorgen so genannte Socket-Dienste dafür, dass die zu sendenden und empfangenen Daten entsprechend dem TCP- bzw. dem UDP-Protokoll verarbeitet werden. Socket heißt übersetzt »Steckdose«. Ein Socket verbindet wie eine Steckdose zwei entfernte Geräte miteinander, ohne dass man wissen muss, wie die Verbindung physikalisch realisiert wird. Unter Windows übernimmt diese Aufgabe die Winsock-Schnittstelle, die ein Teil des Windows API[2] ist. Diese Schnittstelle können Sie direkt über das

1. Die Startsequenznummer wird von TCP nicht vorgeschrieben, ist aber meistens die 1.
2. API = Application Interface. Eine Schnittstelle zu den Funktionen einer Applikation, meist in Form von klassischen DLL-Dateien oder COM-Komponenten. Das Windows-API beinhaltet ca. 1000 Funktionen, die Programme nutzen können, um auf die Funktionalität von Windows zurückzugreifen.

Windows-API nutzen, was allerdings recht komplex ist. Einfacher ist die Verwendung von speziellen Komponenten, die meist Bestandteil einer Programmiersprache sind. Unter Visual Basic 6 nutzen Sie dazu z. B. das Winsock-Steuerelement, unter C#, VB.NET und anderen .NET-Sprachen nutzen Sie die `Socket`-Klasse (die Sie im Namespace `System.Net` finden). Damit können Sie eigene Server und Clients entwickeln, die über TCP oder UDP miteinander kommunizieren, ohne sich um das komplexe Protokoll selbst zu kümmern.

2.7 Die wichtigsten höheren Protokolle

Neben den bereits beschriebenen Protokollen IP, TCP und UDP (ab Seite 41) werden im Internet noch einige weitere Protokolle verwendet. E-Mails werden z. B. über das SMTP-Protokoll versendet, HTML-Seiten über das HTTP-Protokoll. Die für Programmierer wichtigsten Protokolle beschreibe ich in den folgenden Abschnitten. Sie sollten diese Protokolle schon grundlegend kennen. Dann verstehen Sie die Kommunikation zwischen einem Internetserver und einem Client wesentlich besser. Ich muss jedoch zugeben, dass ich bis vor kurzem nur wenig von diesen Protokollen wusste und trotzdem erfolgreich für das Internet programmieren konnte. Ich hab mich aber z. B. immer gefragt, wie der Server den Browsertyp erkennen kann. Jetzt, wo ich weiß, dass der Browser Daten über sich selbst im HTTP-Header zum Server sendet, fällt mir die Antwort leicht.

Der Online-Artikel »Tools zum Testen der Kommunikation über höhere Protokolle« beschreibt, wie Sie verschiedene Tools wie zum Beispiel einen Paket-Sniffer einsetzen, um die Kommunikation zwischen Client und Server testen und überwachen zu können.

2.7.1 Das HTTP-Protokoll

Das HTTP-Protokoll ist wohl das wichtigste Internetprotokoll für Programmierer. Nicht, dass wir Programmierer direkt damit zu tun hätten, wir sollten dieses Protokoll lediglich kennen, um die Kommunikation zwischen Webserver und Client (Browser) besser zu verstehen. Das HTTP-Protokoll besteht zunächst einmal aus einem HTTP-Header und einem optionalen Datenbereich. Der HTTP-Header enthält einen der

HTTP-Befehle (`Get`, `Put`, `Post` etc.) mit Argumenten und weitere Daten, wie z. B. Informationen über den Browser. Der Datenbereich enthält das HTML-Dokument bzw. Daten, die der Client zum Server hochladen will.

Das HTTP-Protokoll wird im RFC 2616 beschrieben.

Eine HTTP-Sitzung besteht aus einem »Request« (der Anforderung) und einem »Response« (der Antwort).

Die Anforderung

In der Anforderung (»Request«) sendet der Client einen der HTTP-Befehle und zusätzliche Informationen im Header zum Server. Der Header enthält in der ersten Zeile den Befehl mit Argumenten und einer Angabe zur verwendeten HTTP-Version. Danach folgen Zeilen, die definieren, welche Grafikdateitypen, Sprachen und Codierungen der Client akzeptiert, und Zeilen mit Informationen über den Client und den Server.

Das folgende Beispiel zeigt eine typische HTTP-Anforderung in einem Intranet. Angefordert wird die Datei »default.htm« im Stammordner des Webservers. Client ist der Internet Explorer 5.0:

```
GET /default.htm HTTP/1.1.
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*.
Accept-Language: de.
Accept-Encoding: gzip, deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0).
Host: Trillian2000p.
Connection: Keep-Alive.
.
```

Ruft der Client ein Dokument ab, das bereits in seinem Cache gespeichert ist, enthält die HTTP-Anforderung normalerweise eine Angabe darüber, dass nur neuere Dokumente zurückgeliefert werden sollen:

```
GET /default.htm HTTP/1.1.
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*.
Accept-Language: de.
Accept-Encoding: gzip, deflate.
```

```
If-Modified-Since: Fri, 27 Jul 2001 11:07:36 GMT.
If-None-Match: "30f488578c16c11:87d".
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0).
Host: Trillian2000p.
Connection: Keep-Alive.
.
```

HTTP 1.1 definiert die in Tabelle 2.1 beschriebenen »Methoden«.

Methode	Bedeutung
GET *URL HTTP-Version*	Anforderung einer Ressource, die in einer URL definiert wird. GET kann mit If-Feldern im Header erweitert werden. If-Modified-Since bewirkt, dass nur neuere Dokumente abgerufen werden.
HEAD *URL HTTP-Version*	HEAD ist zunächst identisch mit GET mit dem Unterschied, dass der Server keine Daten in der Antwort übertragen darf. Damit können Informationen über die angeforderte Ressource ausgelesen werden, ohne die Ressource selbst zu übertragen.
POST *URL HTTP-Version*	POST wird verwendet, um Daten vom Client zum Server zu übertragen, die zu der angegebenen URL gehören. Ein ausgefülltes HTML-Formular wird z. B. über POST an ein Programm auf dem Server übertragen (das in der URL angegeben ist) und dort ausgewertet.
PUT *URL HTTP-Version*	Mit PUT kann ein Client Daten an den Server übertragen, die dieser unter dem in der URL angegebenen Dateinamen speichern soll (vorausgesetzt, der Benutzer besitzt das Schreibrecht auf dem Ordner). FTP-Clients/Server arbeiten z. B. mit dieser Methode.
DELETE *URL HTTP-Version*	Mit dieser Methode kann ein Client eine Datei auf dem Server löschen, sofern das entsprechende Recht vorliegt.

Methode	Bedeutung
TRACE *URL HTTP-Version*	Trace führt dazu, dass der Server die übertragenen Daten wieder zurücksendet. Diese Methode wird verwendet, um in fehlerhaften Systemen herauszufinden, welche Daten der Server wirklich erhält.
CONNECT	Diese spezielle Methode wird mit Proxy-Servern verwendet, die dynamisch zu einem »Tunnel« umgeschaltet werden können (was immer das auch ist ...).

Tabelle 2.1: Die Methoden von HTTP 1.1

Tabelle 2.2 zeigt die wichtigsten Header-Felder für eine Anforderung:

Feld	Bedeutung
Accept	definiert die akzeptierten Medientypen für die Antwort
Accept-Charset	definiert die akzeptierten Zeichensätze für die Antwort
Accept-Encoding	definiert die akzeptierten Verschlüsselungen für die Antwort
Accept-Language	definiert die akzeptierten Sprachen für die Antwort
Authorization	enthält die (verschlüsselten) Logindaten des Benutzers für geschützte Webs.
Expect	enthält Beschreibungen von speziellen Server-Features, die vom Client benötigt werden. Ein Server, der diese Features nicht unterstützt, antwortet mit der HTTP-Fehlermeldung »417 – Expectation failed«.
From	kann verwendet werden, um die E-Mail-Adresse des Benutzers oder spezielle Logging-Informationen an den Server zu übertragen
Host	die Hostadresse und der Port des Servers

Feld	Bedeutung
`If-Modified-Since`	definiert, dass die angeforderte Ressource nur dann übertragen werden soll, wenn diese seit dem angegebenen Datum modifiziert wurde
`If-Unmodified-Since`	wie `If-Modified-Since`, nur negiert
`Proxy-Authorization`	enthält Informationen, die den Client gegenüber einem Proxy identifizieren
`Referer`	definiert die URL des Aufrufers. »Referer« ist übrigens falsch geschrieben und müsste eigentlich »Referrer« heißen.
`User-Agent`	enthält Informationen über den Client wie z. B. den Browsertyp und die Version

Tabelle 2.2: Die wichtigsten Header-Felder einer Anforderung

Die Antwort

Auf eine Anforderung (Request) antwortet der Server mit einem Response. Diese Antwort besteht wieder aus einem Header und einem Datenbereich. Der Header beginnt mit einer Statuszeile, die die HTTP-Version und Informationen über den Erfolg bzw. über Fehler enthält, die bei der Abarbeitung der Anforderung aufgetreten sind. Danach folgen Header-Felder, die z. B. den Server identifizieren und das auf dem Server aktuelle Datum enthalten. Antwortet der Server auf einen `GET`-Request, sieht die Antwort des Servers etwa so aus, falls alles in Ordnung ist:

```
HTTP/1.1 200 OK.
Server: Microsoft-IIS/5.0.
Date: Sun, 29 Jul 2001 15:09:05 GMT.
Content-Type: text/html.
Accept-Ranges: bytes.
Last-Modified: Fri, 27 Jul 2001 11:07:36 GMT.
ETag: "30f488578c16c11:87d".
Content-Length: 375.
.
<html>.
<head>.
```

```
<title>Nitty-Gritty-ASP-Beispielweb</title>.
</head>.
<body bgcolor="silver" >.
<h1>Nitty-Gritty-ASP-Beispielweb - Homepage</h1>.
<p style="font-size:14pt;font-family:fantasy">.
Willkommen im Nitty-Gritty-ASP-Beispielweb.<br>.
Unser <a href=/Shop/default.asp>Shop</a> bietet Lebensmittel aus den
verschiedensten Regionen dieser Welt..
</p>.
</body>.
</html>.
```

Bei irgendwelchen Fehlern antwortet der Server mit einer HTTP-Fehlermeldung (deren HTML-Text Sie in der Administration des IIS selbst definieren können). Die Anforderung einer Datei, die nicht vorhanden ist, wird z. B. mit dem HTTP-Fehler 404 beantwortet:

```
HTTP/1.1 404 Object Not Found.
Server: Microsoft-IIS/5.0.
Date: Sun, 29 Jul 2001 15:14:55 GMT.
Connection: close.
Content-Length: 3238.
Content-Type: text/html.
.
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">.
<html dir=ltr>.
.
<head>.
...
```

Der hier nicht dargestellte Teil der Antwort enthält den Rest des HTML-Dokuments (das bei »`<!DOCTYPE`« beginnt).

Bei einer Anforderung mit dem `If-Modified-Since`-Feld antwortet der Server mit dem Status »304 – Not Modified«, wenn die Ressource nicht geändert wurde:

```
HTTP/1.1 304 Not Modified.
Server: Microsoft-IIS/5.0.
Date: Sun, 29 Jul 2001 15:20:34 GMT.
Content-Location: http://Trillian2000p/Default.htm.
```

```
ETag: "30f488578c16c11:87d".
Content-Length: 0.
.
```

Tabelle 2.3 zeigt die wichtigsten Statuscodes, Tabelle 2.4 die wichtigsten Response-Felder.

Statuscode	Text	Bedeutung
200	OK	eine Anforderung, eine Ressource zu senden, wurde erfolgreich verarbeitet. Die angeforderte Ressource befindet sich im Datenbereich.
201	Created	eine Anforderung, eine Ressource zu erstellen, wurde erfolgreich verarbeitet
202	Accepted	eine Anforderung, einen Prozess auszuführen, wurde erfolgreich verarbeitet. Der Prozess kann allerdings zum Zeitpunkt des Response noch laufen.
302	Object moved	die angeforderte Ressource wurde verschoben. Wird zurückgegeben, wenn die angeforderte Ressource eine Weiterleitung (Redirect) enthält. Das Feld `Location` enthält die neue URL.
304	Not modified	die angeforderte Ressource wurde (bezogen auf das im Request übertragene Dateidatum) zwischenzeitlich nicht geändert und muss deshalb nicht übertragen werden
400	Bad Request	die Anforderung konnte vom Server aufgrund einer deformierten Syntax nicht verstanden werden
401	Unauthorized	die Anforderung erfordert eine Autorisation des Benutzers, die nicht übergeben wurde
403	Forbidden	der Zugriff auf die Ressource ist abgesichert und kann im aktuellen Benutzerkontext nicht erfolgen

Status-code	Text	Bedeutung
404	Not found	die angeforderte Ressource wurde nicht gefunden
405	Method not allowed	die im Request angegebene Methode ist (zum gegenwärtigen Zeitpunkt) nicht erlaubt. Dieser Fehler wird z. B. dann erzeugt, wenn ein HTML-Formular abgesendet wird und der in Action angegebene Dateiname falsch ist (das muss man erst einmal herausfinden ...).
408	Request Time-out	einige Server-Antworten, wie z. B. »401 – Unauthorized«, ermöglichen dem Client eine erneute Anforderung zu senden. Antwortet der Client nicht in der Zeit, die der Server als Timeout definiert hat, antwortet der Server mit dem Status 408.
505	HTTP Version not supported	die HTTP-Version der Anforderung wird nicht unterstützt

Tabelle 2.3: Die wichtigsten HTTP-Statuscodes

Feld	Bedeutung
`Location`	wird verwendet, um den Client auf eine andere URL umzulenken (Redirect). Der Client erhält eine Antwort mit dem Status »302 _ Object moved« mit der Angabe der URL der Ressource, auf die umgeleitet wurde, im Feld `Location`. Der Client reagiert normalerweise darauf, indem er die Ressource über die neue URL neu anfordert.
`Server`	enthält Informationen über den Server
`WWW-Authenticate`	wird mit dem Status 401 (Access denied) gesendet und definiert die möglichen Authentifizierungs-Methoden für den Client

Tabelle 2.4: Die wichtigsten HTTP-Felder eines Response

2.7.2 Das SMTP-Protokoll

Über das SMTP-Protokoll werden E-Mails zwischen Client und SMPT-Server und zwischen SMPT-Servern selbst versendet. Das SMTP-Protokoll ist weniger komplex als HTTP, definiert aber dennoch viele Möglichkeiten. Ich zeige hier nur die Grundlagen von SMTP auf. Der RFC 2821 beschreibt dieses Protokoll ausführlich.

Eine SMTP-Sitzung besteht aus einigen Client-Anforderungen und Antworten des Servers. Der Client baut zunächst eine Verbindung zum SMTP-Server über TCP/IP auf, dieser antwortet mit einer Ready-Meldung, der Client sendet dann HELO mit der Hostadresse des Ziels. Der Server antwortet mit einer Statusmeldung etc. Im Verlauf dieses Austausches von Daten wird dann auch die E-Mail gesendet. Eine typische SMTP-Sitzung sieht z. B. so aus:

```
Server: 220 fwd05.sul.t-online.com T-Online ESMTP receiver
     ➔ fsmtpd ready.
Client: HELO t-online.de
Server: 250 Ok.
Client: MAIL FROM: juergen.bayer@addison-wesley.de
Server: 250 Ok.
Client: RCPT TO: zaphod@galaxy.com
Server: 250 Ok.
Client: DATA
Server: 354 Ok, start with data.
Client: DATE: Sun. 01 Jul 2001 21:15:22 +1000
        From: juergen.bayer@addison-wesley.de
        To: zaphod@galaxy.com
        Subject: Party?

        Hey Zaphod, hast Du Lust zu einer Party im
        Restaurant am Ende des Universums?
        .
Server: 250 Message accepted.
Client: QUIT
Server: 221 fwd05.sul.t-online.com closing.
```

(der Pfeil steht dafür, dass die Zeile aus Formatierungsgründen umbrochen wurde)

Achten Sie besonders darauf, dass die eigentliche Mail mit einem Punkt abgeschlossen wird, der in einer eigenen Zeile steht. Daran erkennt der Server, dass die Mail zu Ende ist.

3 Der Internet Information Server

Der Internet Information Server (IIS) ist, wie Sie ja bereits wissen, der Webserver von Microsoft. Dieses Kapitel gibt zunächst einen Überblick über die Komponenten des IIS und zeigt dann, wie Sie den IIS administrieren.

Das für den Real-World-Einsatz des IIS wichtige Thema »Sicherheit« behandelt der Online-Artikel »IIS-Sicherheit«, den Sie auf der Website zum Buch finden.

3.1 Die Komponenten des IIS

Der Internet Information Server ist nicht nur ein Webserver, der ASP-Programme ausführen kann. Er integriert noch einige weitere Komponenten, die teilweise für Programmierer wichtig sind und die ich deshalb hier kurz beschreibe. Einige Komponenten, wie der FTP-Server, sind direkt im IIS integriert, andere, wie der Microsoft Transaction Server, sind separate Anwendungen, die nicht ausschließlich vom IIS verwendet werden. Ich beschreibe hier nur die Komponenten, die in einem direkten Zusammenhang mit dem IIS stehen.

Der FTP-Server

Der im IIS integrierte FTP-Server funktioniert sehr einfach. Alle Dateien und Ordner, die Sie im Basisverzeichnis des FTP-Servers ablegen (normalerweise ist das der Ordner \INETPUB\WWWROOT), kann ein FTP-Client oder ein Webbrowser über FTP abrufen. Die Dateien auf Ihrem System können Sie z. B. im Internet Explorer über die URL *ftp://localhost* abrufen.

In der Administration des FTP-Servers legen Sie fest, ob ein anonymer Zugriff möglich ist. Lassen Sie diesen zu, kann jeder Benutzer auf die Dateien zugreifen. Lassen Sie den anonymen Zugriff nicht zu, haben nur Benutzer Zugriff, die als normale Windows-Benutzer registriert sind und unter Windows einen passenden Zugriff auf die entsprechende Datei besitzen.

Der NNTP-Server

Der NNTP-Server ist ein einfacher Newsserver, über den Sie eigene Newsgroups einrichten können. Dieser Server steht nur unter Windows NT Server und Windows 2000 Server zur Verfügung.

Der virtuelle SMTP-Server

Der virtuelle SMTP-Server ermöglicht Ihnen das Senden von E-Mails in beliebigen Programmen. Die E-Mail wird dazu entweder in Textform im SMTP-Protokoll in ein spezielles Verzeichnis abgelegt (womit sie automatisch versendet wird) oder mit Hilfe von COM-Objekten versendet. Der SMTP-Server übernimmt die Weiterleitung an den Empfänger. Dazu ist nichts weiter als ein vorhandener Internetzugang notwendig. Im Online-Artikel »ASP-Tipps und Tricks« beschreibe ich, wie Sie mit Hilfe des virtuellen SMTP-Servers (oder mit Hilfe eines entfernten SMPT-Servers) Mails versenden.

Die FrontPage-Servererweiterungen

Die FrontPage-Servererweiterungen sind spezielle Komponenten, die auf dem Webserver installiert werden. Diese Komponenten ermöglichen externen Programmen wie Microsoft FrontPage und Microsoft Visual Studio den direkten Zugriff auf die Dateien einer Website. Ein Entwickler kann eine Website damit auch von entfernten Rechnern aus in einer passenden Entwicklungsumgebung bearbeiten.

Die ISAPI-Schnittstelle

Die Internet Server API (ISAPI) ist eine Schnittstelle, die Programmierer nutzen können, um mit beliebigen Programmiersprachen Internetanwendungen zu entwickeln. Die Anwendung wird dazu in Form einer DLL kompiliert. Diese DLL exportiert einige durch den ISAPI-Standard festgelegte Funktionen. Das Programm nutzt die API-DLLs der ISAPI mit speziellen Internetfunktionen. Mit der ISAPI-Schnittstelle können Sie ISAPI-Erweiterungen und ISAPI-Filter erzeugen. ISAPI-Erweiterungen werden einfach über eine URL aufgerufen und enthalten Internetprogramme, die – wie ASP – auch HTML-Seiten erzeugen können. ISAPI-Filter sind intern mit bestimmten einfachen Ereignissen verknüpft, die beim Abrufen eines Webdokuments auftreten. Dazu gehören z. B. Ereignisse wie das Verarbeiten der

HTTP-Header einer Anforderung und die Authentifizierung des Clients. Ein ISAPI-Filter wird automatisch vom Webserver aufgerufen, wenn eine Webanforderung das Ereignis auslöst, für das der Filter registriert ist.

ISAPI-Erweiterungen sind mittlerweile veraltet, da ASP und vor allen Dingen ASP.NET wesentlich einfacher zu programmieren sind und mehr Möglichkeiten bieten. ISAPI-Filter sind u. U. noch sinnvoll, da diese Filter beliebige Anforderungen automatisch bearbeiten können.

Weitere Informationen zu ISAPI finden Sie in der Online-Dokumentation des IIS (*http://localhost/iisHelp/iis/misc/default.asp*), indem Sie »ISAPI« als Suchbegriff eingeben, oder im MSDN (*http://msdn.microsoft.com/library*), indem Sie nach »Developing ISAPI Extensions» und »Developing ISAPI Filters« suchen.

Der Zertifizierungsserver

Der Zertifizierungsserver (Microsoft Certificate Server) erstellt und verwaltet digitale Zertifikate. Digitale Zertifikate bestätigen die Echtheit von Unternehmen und Einzelpersonen im Internet. Diese Zertifikate werden für viele abgesicherte Anwendungen im Internet benötigt. Der sichere Zugriff auf Webseiten über SSL (Secure Sockets Layer) ist z. B. nur mit gültigen Zertifikaten möglich. Der Person oder dem Unternehmen wird dazu ein öffentlicher Schlüssel zugeordnet, der im Zertifikat zusammen mit dem Namen der Person bzw. des Unternehmens enthalten ist. Abgesicherte Dokumente werden mit dem Zertifikat signiert, womit der Empfänger überprüfen kann, ob das Dokument wirklich vom Sender kommt. Der Empfänger muss dazu lediglich den öffentlichen Schlüssel des Senders kennen.

Der Zertifizierungsserver steht nur unter Windows NT Server und Windows 2000 Server zur Verfügung.

Der Microsoft Index Server

Der Index Server, der nur unter Windows NT Server und Windows 2000 Server zur Verfügung steht, ermöglicht die Volltextsuche in beliebigen Dateien. Mit Hilfe dieses Servers können Sie mit wenig Programmieraufwand z. B. eine Suchseite für Ihr Web einrichten. Auf der

Website zum Buch finden Sie einen Artikel, der die Programmierung mit dem Index Server beschreibt.

Der Microsoft Transaction Server (MTS) bzw. COM+

Der Microsoft Transaction Server (MTS) ist unter Windows NT 4 ein separater Server, den Sie mit dem Windows NT Option Pack installieren können. In Windows 2000 sind die Dienste, die der MTS unter Windows NT anbietet, in Form der so genannten Komponentendienste (Component Services) bereits in COM+ integriert. Das Component Object Model (COM) stellt, wie Sie ja sicherlich bereits wissen, eine Infrastruktur zur Verfügung, die ermöglicht, dass Anwendungen Objekte verwenden können, deren Klassen in separaten Dateien (so genannten COM-Komponenten) im binärer Form gespeichert sind. Der MTS bzw. die Komponentendienste von COM+ erweitern die COM-Infrastruktur um wichtige Features wie ein flexibles und ressourcenschonendes Komponentenmanagement und das Handling von Transaktionen[1]. Um einem weit verbreiteten Irrtum direkt vorzubeugen: Der MTS ist nicht ausschließlich dazu da, Transaktionen zu handeln. Eine weitere, von Transaktionen zunächst unabhängige Aufgabe des MTS ist, die auf einem System installierten Komponenten so zu verwalten, dass möglichst wenig Ressourcen verbraucht werden und dass das Instanzieren von Objekten aus diesen Komponenten möglichst performant ist.

Komponenten können von normalen Anwendungen und von Webanwendungen verwendet werden. Besonders für Webanwendungen in klassischen ASP-Seiten spielen solche Komponenten eine große Rolle. Eine ASP-Seite, die für das eigentliche Programm eine Komponente verwendet, ist wesentlich schneller als eine ASP-Seite, die das Programm selbst enthält. Dummerweise erzeugt jeder Aufruf einer ASP-Seite eine neue Instanz der Komponente oder zumindest einer Klasse dieser Komponente. Greifen sehr viele Benutzer gleichzeitig

1. Als Transaktion wird in der Computerwelt eine datenverändernde Aktion bezeichnet. Werden innerhalb der Aktion mehrere unterschiedliche Datensätze (in einer oder mehreren Datenbanken) verändert, stellt eine Transaktion sicher, dass entweder alle oder – im Fehlerfall – gar keine Veränderungen stattfinden.

auf die ASP-Seite zu, sind die Ressourcen des Rechners schnell ausgeschöpft. Ist die Komponente allerdings im MTS bzw. in COM+ registriert, werden die Ressourcen massiv geschont. Der MTS bzw. COM+ erzeugt dazu nur eine bestimmte Maximalanzahl an Komponenten und verteilt die verfügbaren dynamisch an die einzelnen Clients (in unserem Fall an die ASP-Seiten). Komplexe Techniken ermöglichen dabei eine optimale Performance.

Der MTS bzw. COM+ ist für Internetprogrammierer wichtig, die große Websites entwickeln, auf die gleichzeitig sehr viele Benutzer zugreifen. Sie finden auf der Website zum Buch einen Artikel, der die Arbeit mit dem MTS bzw. COM+ beschreibt.

3.2 IIS-Administration für Programmierer

Der IIS kann über mehrere Programme administriert werden: über ein Snap-In in der Microsoft Management Console (MMC), über ASP-Programme (bis Windows NT 4) oder über den Personal Web Manager. Die MMC bietet die besten Möglichkeiten und zeigt vor allen Dingen alle Optionen, was bei den anderen Administrations-Anwendungen nicht der Fall ist. Der Personal Web Manager führt z. B. häufig zu Verwirrungen. Dieses Programm bietet nur sehr rudimentäre Administrations-Features. In den Eigenschaften eines Webordners fehlen dort die meisten Optionen. Ich konzentriere mich deshalb auf die MMC.

Die MMC-Administration

Die IIS-MMC-Administration befindet sich in der Datei IIS.MSC, die Sie unter Windows NT und Windows 2000 im Ordner \WINNT\SYSTEM32\INETSRV finden. Ein Doppelklick auf diese Datei öffnet die MMC. Sie können diese Datei aber auch anders öffnen: Wählen Sie dazu unter Windows NT START | PROGRAMME | WINDOWS NT OPTION PACK | MICROSOFT INTERNET INFORMATION SERVER | INTERNETDIENSTE-MANAGER. Unter Windows 2000 öffnen Sie den Ordner VERWALTUNG in der Systemsteuerung und starten dort den INTERNETDIENST-MANAGER. Die Administration sieht unter den verschiedenen Windows-Versionen ähnlich aus. Ich beschreibe deshalb nur die neuere Version (IIS 5 unter Windows 2000). Bei gravierenden Unterschieden zum IIS 4 weise ich darauf hin.

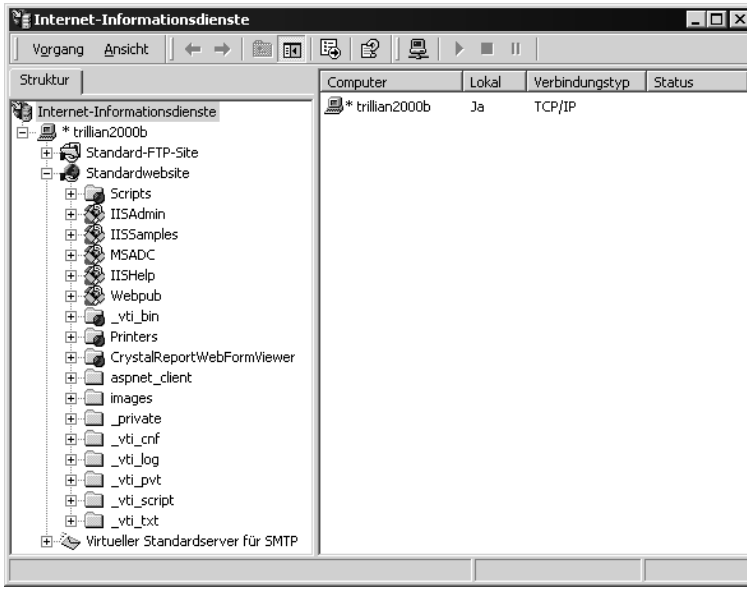


Bild 3.1: Das Snap-In zur IIS-Administration in der MMC

Abbildung 3.1 zeigt die IIS-Administration in der MMC mit bereits aufgeklapptem Ordner für die Standardwebsite[1]. Ordner mit einem einfachen Ordnersymbol bezeichnen Dateiordner, die im Basisverzeichnis des IIS gespeichert sind. Alle Ordner, die Sie im Basisverzeichnis anlegen, werden automatisch in der IIS-Administration angezeigt (dazu müssen Sie allerdings die Ansicht über das Kontextmenü des Standardwebsite-Ordners aktualisieren). Ordner mit einer Weltkugel bezeichnen virtuelle Ordner, deren korrespondierende Dateiordner beliebig auf den Systemlaufwerken gespeichert sein können. Ordner, die mit dem Symbol gekennzeichnet sind, das wie ein

1. Der IIS kann mehrere Websites verwalten. Jeder Website wird dazu eine separate IP-Adresse oder ein anderer Port zugeordnet. Der IIS 4 nutzt diese Technik für eine spezielle Administrations-Website, auf die nur bestimmten Rechnern Zugriff gegeben wird (per Default nur dem lokalen Rechner). Unter dem IIS 5 scheint die Möglichkeit, mehrere Websites einrichten zu können, keine Rolle mehr zu spielen.

geöffneter Kasten aussieht, laufen als Webanwendung. Ein solcher Ordner ermöglicht, dass die darin laufenden ASP-Programme Daten für einzelne Sitzungen oder für die gesamte Anwendung zwischenspeichern (was mit normalen Ordnern nicht geht) und erlaubt das Debuggen. Ab Seite 64 gehe ich noch näher auf Webanwendungen ein. Ob der Ordner im Basisverzeichnis gespeichert ist oder ein virtueller Ordner ist, können Sie am Symbol nicht erkennen.

3.2.1 Das Basisverzeichnis

Der IIS verwaltet seine Webdateien primär in einem Basisverzeichnis, das in den Eigenschaften des Webservers eingestellt werden kann. Normalerweise ist dies der Ordner \INETPUB\WWWROOT direkt unter dem Stammordner des Systemlaufwerks. Dieser Ordner ist die physikalische Repräsentanz des virtuellen Root-Ordners. Alle Dateien in diesem und allen Unterordnern können über relative Pfandangaben mit einem Webbrowser abgerufen werden, sofern dies nicht durch Sicherheitseinstellungen unterbunden wird. In der Administration des IIS können Sie aber auch beliebige andere Ordner auf neue virtuelle Webordner mappen. Dort geben Sie auch an, welchen Zugriff Clients auf die einzelnen Ordner oder Dateien besitzen (LESEN, SCHREIBEN, AUSFÜHREN etc.).

3.2.2 Eigenschaften der Standardwebsite

Wenn Sie auf dem Ordner der Standardwebsite mit der rechten Maustaste klicken, können Sie im Kontextmenü die Eigenschaften dieser Website öffnen.

Übersicht

Zur Übersicht beschreibt Tabelle 3.1 zunächst die einzelnen Register. Einige Register werden auf den folgenden Seiten noch näher erläutert.

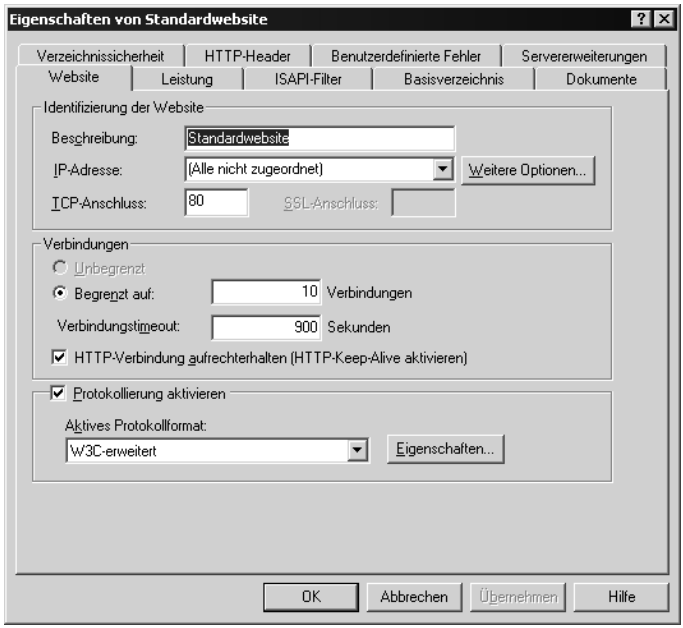


Bild 3.2: Die Eigenschaften der Standardwebsite

Register	Beschreibung
Website	In Website-Register (Bild 3.2) stellen Sie die Grundeinstellungen der Website, wie z. B. deren Beschreibung und den Timeout für Verbindungen, ein. Diese Einstellungen beschreibe ich im folgenden Abschnitt.
Leistung	Im Leistung-Register können Sie die Leistung der Website optimieren und einschränken. So können Sie z. B. die Netzwerkbandbreite auf einen bestimmten Wert reduzieren, damit Ihr Netz durch den IIS nicht überlastet wird. Eine Einschränkung der CPU-Verwendung ist u. U. dann sinnvoll, wenn auf dem Rechner noch weitere Server laufen.
ISAPI-Filter	In diesem Register binden Sie ISAPI-Filter ein.

Register	Beschreibung
Basisverzeichnis	Hier stellen Sie die Grunddaten des Ordners ein, in dem die Dateien und Unterordner des Webs verwaltet werden. Dazu gehören das physikalische Verzeichnis und die Rechte, die Internetanwender auf diesem Ordner besitzen (Lesen, Schreiben, Ausführen etc.). Für Unterordner (die dieses Register auch besitzen) konfigurieren Sie hier ebenfalls die Einstellungen einer eventuellen Webanwendung. Webanwendungen beschreibe ich ab Seite 66.
Dokumente	In diesem Register stellen Sie ein, welches Dokument als Standard-Dokument verwendet wird, wenn der Anwender keine Dateiangabe in die URL integriert (also nur eine Pfadangabe macht). Außerdem können Sie hier ein HTML-Dokument angeben, das als Fußzeile für alle Dokumente des Web verwendet wird. Solche Fußzeilen sind aber nicht zu empfehlen. Zum einen wird die Zeile immer unter dem eigentlichen Dokument angezeigt und nicht am Ende des Webbrowser-Fensters, zum anderen wird die Fußzeile nicht angezeigt, wenn der Anwender keinen Dateinamen in die URL integriert und folglich das Standarddokument verwendet wird.
Verzeichnissicherheit	Hier können Sie u. a. einstellen, ob Benutzer sich in das System einloggen müssen, wenn sie versuchen ein Dokument abzurufen. Außerdem können Sie den Zugriff auf bestimmte IP-Adressen oder Domänennamen beschränken (nur Windows NT/2000 Server) und die sicherer Kommunikation über SSL mit Server-Zertifikaten einrichten.
HTTP-Header	In diesem Register können Sie festlegen, welche Werte im Header einer HTML-Seite an den Client zurückgegeben werden. Der HTTP-Header wird genutzt, um Informationen zwischen Client und Server auszutauschen. Der Client überträgt z. B. Informationen über die Art und Version des Webbrowsers im Header.

Register	Beschreibung
Benutzerdefinierte Fehler	Das HTTP-Protokoll definiert einige Fehler, die mit Nummern bezeichnet sind. Der Fehler 404 wird z. B. erzeugt, wenn ein Benutzer einen unbekannten Dateinamen in der URL verwendet. Für diese Fehler können Sie benutzerdefinierte Fehlermeldungen angeben. Voreingestellt sind die Standard-Fehlermeldungen des IIS. Wenn Sie für Ihr Web bessere oder schönere Fehlermeldungen ausgeben wollen, erzeugen Sie eine HTML-Datei für den entsprechenden Fehler und geben diese als Fehlerdatei an. Sie können dabei ja die voreingestellten HTML-Dateien als Basis verwenden. In einem Web ist eine Anpassung dieser Seiten schon deshalb wichtig, um Angaben zum technischen Support in die Fehlermeldung integrieren zu können. Sie finden die Default-Fehlerdokumente in \WINNT\HELP\IISHELP\COMMON\.
Servererweiterungen	In diesem Register können Sie die FrontPage-Servererweiterungen grundlegend administrieren. Hier können Sie z. B. festlegen, ob die Erstellung von Dokumenten im Web (über ein kompatibles Programm) möglich ist. Die eigentliche Administration der FrontPage-Servererweiterungen ist übrigens eine MMC-Snapin, das finden Sie bei Windows 2000 unter SYSTEMSTEUERUNG	VERWALTUNG	ADMINISTRATOR FÜR SERVERERWEITERUNGEN. Die Administration der FrontPage-Servererweiterungen wird in diesem Buch nicht weiter behandelt.

Tabelle 3.1: Die Register der Eigenschaften einer Website

Das Register »Website«

In Website-Register (Abbildung 3.2) stellen Sie die Grundeinstellungen der Website, wie z. B. deren Beschreibung und den Timeout für Verbindungen, ein:

- **Beschreibung:** Eine einfache Beschreibung der Website für interne Zwecke.

- **IP-Adresse und TCP-Anschluss:** Falls auf dem Server mehrere Websites installiert sind, muss jeder dieser Sites eine separate IP-Adresse[1] oder – bei gleicher Adresse – ein separater Port zugeordnet werden. Die Einstellung (ALLE NICHT ZUGEORDNET) bedeutet, dass diese Website über alle dem Rechner zugeordneten IP-Adressen aufgerufen werden kann. In der Einstellung TCP-Anschluss stellen Sie den Port ein, an dem die Website auf Anforderungen reagiert. Normale Websites laufen unter dem Port 80. Spezielle Websites, z. B. eine Website für Administrationszwecke, laufen meist unter anderen Portnummern.
- **Verbindungen:** Hier stellen Sie die Anzahl der gleichzeitig möglichen Verbindungen und das Verbindungs-Timeout ein. Der IIS kann unter Windows NT Server und Windows 2000 Server theoretisch unendlich viele Verbindungen bearbeiten (unter den anderen Windows-Versionen sind nur maximal 10 Verbindungen möglich). Zu viele gleichzeitige Verbindungen begrenzen aber die Performance. Wenn Sie die Performance Ihres Web erhöhen wollen, begrenzen Sie die Anzahl der gleichzeitig möglichen Verbindungen. Ein Webbrowser, der versucht eine Verbindung aufzubauen, erhält die HTTP-Meldung 403.9 »Zugriff verboten: Zu viele Benutzer verbunden«, wenn die Anzahl der möglichen Verbindungen damit überschritten würde.

  Der Verbindungs-Timeout regelt die Zeit, die eine Verbindung noch geöffnet bleibt, wenn ein Client sich zwischenzeitlich nicht meldet. Ist das Timeout abgelaufen, wird die Verbindung getrennt und steht damit anderen Clients zur Verfügung. Der Timeout hat allerdings nur dann Auswirkungen, wenn die Anzahl der Verbindungen begrenzt ist. Stehen noch genügend freie Verbindungen zur Verfügung, vergibt der IIS dem Client einfach eine neue Verbindung, wenn dieser sich zurückmeldet.
- **Protokollierung:** Der IIS ist in der Lage, jeden Zugriff auf die Website zu protokollieren. Er verwendet dazu einfach Textdateien. Webmaster können an Hand dieser Dateien Rückschlüsse auf die Webzugriffe ableiten. In den Eigenschaften können Sie einstellen, ob die Protokollierung aktiviert ist und wie die Dateien verwaltet

1. Sie können einem Rechner auch mehrere IP-Adressen zuordnen.

werden. Per Voreinstellung erzeugt der IIS pro Tag eine separate Datei, die Sie im Ordner \WINNT\system32\LogFiles\W3SVC1 finden.

Eigenschaften der Webordner

Jeder Webordner besitzt ähnliche Eigenschaften wie der Stammordner. Lediglich die Register Website, Leistung, ISAPI-Filter und Servererweiterungen fehlen. Für die restlichen Einstellungen können Sie für jedes Unterweb Einstellungen vornehmen, die von der Konfiguration des Stammordners abweichen. So können Sie z. B. den Zugriff auf Unterweb unabhängig vom Stammweb auf bestimmte Benutzer einschränken.

3.2.3 Webanwendungen

Den Stammordner und jeden Unterordner Ihres Web können Sie als »Anwendung« definieren. Damit ermöglichen Sie Sitzungen, die Verwaltung von anwendungs- oder sitzungsglobalen Daten, das Debuggen von ASP-Dokumenten und die Zuordnung von Dateitypen zu ISAPI-Anwendungen.

Die Bezeichnung »Anwendung« ist für diesen Zweck etwas irreführend. Es handelt sich ja schließlich nicht um ein Programm, das von Anwendern benutzt wird, sondern um eine spezielle Konfiguration eines Webordners.

Wenn Sie einen Webordner nicht als Anwendung konfigurieren, können Sie keine globalen Daten in Ihren ASP-Dokumenten verwenden. Erst eine Anwendung ermöglicht die Verwaltung globaler Daten für die gesamte Anwendung oder nur für die einzelnen Sitzungen. Deshalb ist die Bezeichnung »Anwendung« wohl angebracht.

Sie erreichen die Konfiguration für Webanwendungen über das Register Verzeichnis in den Eigenschaften eines Webordners. Im Bereich Anwendungseinstellung können Sie die Anwendung konfigurieren. Falls noch keine Anwendung besteht, können Sie eine über den `Erstellen`-Schalter erzeugen. Über den `Konfigurieren`-Schalter erreichen Sie die Konfiguration (was auch sonst ...).

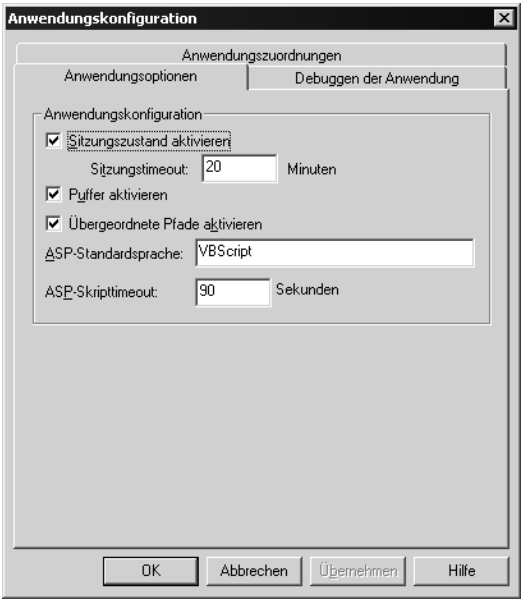


Bild 3.3: Die Konfiguration einer Webanwendung

Sitzungen

Wenn Sie für eine Webanwendung Sitzungen ermöglichen, wird jede Verbindung eines Clients als eine Sitzung betrachtet. Der Sinn solcher Sitzungen ist das Speichern globaler Daten. Damit können Sie auf einfache Weise Daten zwischen einzelnen ASP-Dokumenten austauschen. Für einen Online-Shop müssen Sie sich z. B. die ID des Anwenders merken, um den Warenkorb identifizieren zu können. Diese ID wird üblicherweise in einer Sitzungsvariable verwaltet. In der Konfiguration können Sie zudem das Timeout einer Sitzung einstellen. Die Voreinstellung ist 20 Minuten. Ruft der Anwender innerhalb dieser Zeit kein Dokument ab, wird die Sitzung beendet. Die globalen Sitzungsdaten gehen dann verloren. Das Timeout der Sitzung kann auch in ASP über das `Session`-Objekt angepasst werden.

Globale Daten

In ASP können Sie globale Daten für eine Anwendung oder eine Sitzung speichern. Diese Daten können dann grundsätzlich von allen ASP-Dokumenten innerhalb der Anwendung gesetzt und abgefragt werden. Unterschieden werden dabei anwendungs- und sitzungsglobale Daten. Anwendungsglobale Daten existieren nur einmal. Wenn eine Sitzung die Daten ändert, ist die Änderung sofort in allen anderen Sitzungen verfügbar. Sitzungsglobale Daten werden für jede Sitzung separat gespeichert. Eine Änderung in einer Sitzung betrifft andere Sitzungen nicht. Das Kapitel 8 zeigt, wie Sie in ASP mit den Objekten `Application` und `Session` globale Daten speichern.

Debuggen

Im Register DEBUGGEN DER ANWENDUNG ermöglichen Sie das Debuggen von ASP-Skripten. Damit ist es möglich, Fehler, die in einem ASP-Dokument auftreten, zu lokalisieren. Das Debuggen von ASP-Skripten wird in Kapitel 7 behandelt, das von ASP.NET-Seiten in Kapitel 11.

Anwendungszuordnungen

Im gleichnamigen Register stellen Sie die Zuordnung von Dateitypen zu ISAPI-Anwendungen ein. Voreingestellt sind die Dateitypen, mit denen der IIS bereits umgehen kann. Dazu gehört auch ASP. Wenn Sie selbst eine ISAPI-Anwendung entwickelt oder eine gekauft haben, können Sie diese Anwendung hier mit einem Dateityp verbinden. Ruft der Benutzer eine Datei mit der entsprechenden Endung ab, wird die Datei über die ISAPI-Anwendung gesendet und dort verarbeitet.

4 HTML

Wenn Sie Webanwendungen entwickeln, müssen Sie HTML kennen. Moderne HTML-Editoren wie Macromedia Dreamweaver, Adobe Go Live oder Microsoft FrontPage ermöglichen Ihnen zwar das Erzeugen von HTML-Seiten, ohne dass Sie eine einzige Zeile Quelltext schreiben. Spätestens aber, wenn Sie mit JavaScript auf der Clientseite oder mit ASP auf der Serverseite Programme entwickeln (die normalerweise HTML-Quellcode erzeugen), werden Sie grundsätzliche HTML-Kenntnisse benötigen. Dieses Kapitel beschreibt deshalb die für Programmierer wichtigen Grundlagen.

4.1 Grundlagen

4.1.1 Was ist HTML?

HTML ist eine reine Dokumentbeschreibungssprache. Mit HTML können Sie nicht programmieren, Sie können aber ein Dokument erzeugen und gestalten. Ein reines HTML-Dokument (ohne CSS, DHTML und JavaScript) ist vergleichbar mit einem einfachen Word-Dokument. Es enthält Texte, Bilder und Links zu anderen Dokumenten. Solche Dokumente kennen Sie ja wahrscheinlich bereits vom Surfen im Internet.

Ein HTML-Dokument wird immer in reinem Textformat gespeichert. In diesem Dokument werden einzelne Inhalte formatiert, Bilder eingebunden und Links erzeugt. Der HTML-Standard, der mittlerweile bei Version 4.0 angelangt ist, definiert dazu so genannte Tags. Ab Seite 69 gehe ich auf die wichtigsten HTML-Tags ein. Zunächst zeige ich aber, wo Sie gute HTML-Referenzen finden, welche Programme Sie zur Erzeugung von HTML-Dokumenten einsetzen können, und schreibe ein wenig über browserspezifische HTML-Erweiterungen.

4.1.2 HTML-Referenzen

Da dieses Buch kein HTML-Buch ist, beschreibe ich lediglich die wichtigsten Tags und Attribute. Wenn Sie sich näher mit HTML beschäftigen wollen, empfehle ich Ihnen das HTML-Buch der Nitty-Gritty-Reihe von Addison-Wesley (ISBN 3-8273-1669-3) und SelfHTML, das beste (deutsche) Online-HTML-Buch, das es gibt. SelfHTML finden Sie im Internet auf der Seite *www.teamone.de/selfhtml*.

Eine ebenfalls sehr gute Online-Referenz ist die der »Web Design Group« (*www.htmlhelp.com/reference/html40/*). Hier finden Sie vor allen Dingen umfassende und übersichtliche Kurzreferenzen zu den einzelnen Tags. Natürlich können Sie auch die offizielle HTML-Referenz verwenden, die im RFC 2616 beschrieben wird. Die Referenz der Web Design Group ist übersichtlich aufgebaut und leichter zu lesen als die offizielle und enthält zusätzliche Hinweise zu Features, die nur von bestimmten Browsern unterstützt werden. Die offizielle Referenz enthält dafür Hinweise zu Elementen, die nicht mehr eingesetzt werden sollten, da sie mittlerweile durch anderen Features ersetzt wurden.

4.1.3 Der HTML-Standard und browserspezifische Erweiterungen

Der HTML-Standard, nach dem sich alle Programme, die mit HTML arbeiten, richten sollten, wird im RFC 2616 beschrieben. Leider werden einige zum Standard gehörende Tags und Attribute von vielen Browsern nicht oder nur unzureichend unterstützt. Dazu gehört zum Beispiel die Möglichkeit, Text in Tabellen an einem bestimmten Zeichen (normalerweise dem Komma) auszurichten. In der HTML-Referenz der Web Design Group (*www.htmlhelp.com/reference/html40/*) finden Sie bei diesen Features eine entsprechende Anmerkung (»... poorly supported by browsers ...«).

Wohl als Ersatz dafür, dass der Standard häufig nicht komplett unterstützt wird, erweitern einige Browser (wie der Internet Explorer und der Netscape) den HTML-Standard um zusätzliche Tags und Attribute, die natürlich nur von den betroffenen Browsern verstanden werden. Ein Beispiel dafür sind die nicht zum Standard gehörenden Attribute `topmargin` und `leftmargin` des `body`-Tag, die beim Internet Explorer die Randbreiten des HTML-Dokuments bestimmen. Der Net-

scape-Browser kann mit diesen Attributen nichts anfangen, kennt aber die Attribute `marginheight` und `marginwidth`, die dasselbe bewirken.

Wenn Sie mit browserspezifischen Erweiterungen arbeiten, ist es natürlich nicht einfach, alle gängigen Browser mit einzubeziehen. Wenn Sie jedoch nur einige Browser als Standard für Ihre Dokumente ansehen (was besonders bei der Entwicklung von Intranet-Webs der Fall sein kann), können Sie einfach die spezifischen Tags und Attribute für diese Browser in Ihr Dokument integrieren. Da Browser unbekannte Tags bzw. Attribute einfach ignorieren, stellt diese Vorgehensweise in den meisen Fällen kein Problem dar. Wenn Sie in einem HTML-Dokument z. B. für den Internet Explorer und für den Netscape die Randbreite angeben wollen, verwenden Sie den `body`-Tag wie folgt:

```
<body topmargin="25" leftmargin="25"
 marginheight="25" marginwidth="25">
```

Browser, die diese Attribute nicht verstehen, werden dann einfach die Standardwerte verwenden. Wenn Sie browserspezifische Tags und Attribute bisher noch nicht kannten, verstehen Sie jetzt, warum auf manchen HTML-Seiten der Hinweis »Optimiert für Netscape« bzw. »Optimiert für den Internet Explorer« angebracht ist.

Für den Internet Explorer und den Netscape kennzeichne ich browserspezifische Elemente in diesem Buch mit »IE« (Internet Explorer) bzw. »NS« (Netscape).

4.1.4 Elemente, Tags und Attribute

HTML-Dokumente bestehen aus reinem Text und/oder so genannten »Elementen«. Ein Element besteht aus einem Start-Tag, einem optionalen Inhalt und einem Ende-Tag. Tags sorgen für eine Formatierung des Element-Inhalts, für das Einbinden von Grafiken, für das Erzeugen von Verweisen (Links) und anderes. Der Start-Tag besteht immer aus umschließenden spitzen Klammern, dem Tag-Bezeichner und optionalen Attributen.

Bei der Auswertung von Tag-Bezeichnern, Attributen und deren Werten unterscheidet HTML 4 Groß- und Kleinschreibung nicht. XML und XHTML (der kommende HTML-Standard) fordern aber wohlgeformte Dokumente. Eine Voraussetzung wohlgeformter Dokumente ist, dass Tags und Attribute immer mit kleinen Buchstaben geschrieben werden müssen. Um Ihre HTML-Dokumente für den neuen Standard vorzubereiten, sollten Sie Tags und Attribute also schon jetzt grundsätzlich kleinschreiben.

Der Ende-Tag sieht prinzipiell aus wie der Start-Tag, nur dass er keine Attribute besitzt und dass dem Tagbezeichner ein Slash vorangestellt wird. Der `b`-Tag, der für eine fette Auszeichnung des Element-Inhalts steht, wird z. B. mit `<b>` begonnen und mit `</b>` abgeschlossen. Daran erkennt der Browser, welcher Text fett dargestellt werden soll und welcher nicht:

```
<b>Das ist voll fett.</b> Hier ist jetzt nichts mehr fett.
```

In diesem Beispiel ist »`<b>Das ist voll fett.</b>`« ein HTML-Element und »`Hier ist jetzt nichts mehr fett.`« reiner Text.

Einige wenige Tags, wie z. B. der `br`-Tag (Zeilenumbruch), werden allerdings in HTML 4 nicht abgeschlossen, weil sie keinen Inhalt besitzen:

```
Das ist Zeile 1<br>
Das ist Zeile 2<br>
```

Einige Tags, die normalerweise abgeschlossen werden müssen, können oft auch ohne abschließenden Tag verwendet werden. Sehr häufig wird dies z. B. beim `p`-Tag (Absatz) angewendet. Um wohlgeformte Dokumente zu erzeugen, sollten Sie diese Tags jedoch auch abschließen.

Tag-Attribute

Viele Tags können Sie mit Attributen feineinstellen. Einige Tags, wie z. B. `<font>`, machen sogar nur Sinn, wenn Attribute verwendet werden. Betrachten Sie ein HTML-Element (also den Tag und den Inhalt) als Objekt (genau das macht DHTML), sind die Attribute Eigenschaf-

ten des Objekts. Für den font-Tag können Sie z. B. immer die Attribute color, face und size einstellen (der Netscape-Browser erlaubt zudem das Attribut point-size).

Attribute werden mit Leerzeichen getrennt im Start-Tag angegeben. Nach dem Namen des Attributs folgt der Zuweisungsoperator (=) und der Wert, der dem Attribut zugewiesen werden soll. Der Wert sollte immer in Anführungszeichen eingefügt werden. Die aktuellen Browser lassen Wertzuweisungen zwar auch ohne Anführungszeichen zu, wenn der Wert nur Zahlen, Buchstaben und/oder den Unterstrich enthält. Für wohlgeformte Dokumente sind Anführungszeichen jedoch vorgeschrieben.

Mehrere Attribute werden einfach durch Leerzeichen getrennt. Das folgende Beispiel verwendet alle Attribute des font-Tag zur Einstellung der Schriftart:

```
<font face="arial" size="12" color="red">
 Das ist roter Text in Arial 12.
</font>
```

Boolsche Attribute

Einige Attribute speichern boolsche Werte. Dazu gehört z. B. das selected-Attribut eines Eintrags in einer Combobox:

```
<select name="car">
  <option selected="selected">
    Ford Puma
  </option>
  <option>
    Porsche 911
  </option>
</select>
```

Sind boolsche Attribute nicht angegeben, verwendet der Browser den Wert False. Um True einzutragen, sollten Sie wie im obigen Beispiel als Wert den Namen des Attributs eintragen. Dann entspricht der Quellcode einem wohlgeformten Dokument. In HTML 4 können Sie alternativ auch nur den Attributnamen schreiben (diese Form wird als »Minimized Boolean« bezeichnet):

```
<option selected>
```

Beachten Sie, dass Ihr Dokument dann aber nicht mehr wohlgeformt ist.

Tags kombinieren

Sie können verschiedene Tags miteinander kombinieren. Wenn Sie einen Text fett, kursiv und mit der Schriftgröße 12 formatieren wollen, verwenden Sie eine Kombination von `b`-, `i`- und `font`-Tags:

```
<b><i>
<font face="arial" size="12" color="red">
 Das ist fetter, kursiver, roter Text in Arial 12.
</font>
</i></b>
```

Zur Erzeugung wohlgeformter Dokumente sollten Sie geschachtelte Tags in der logischen Reihenfolge wieder schließen, so wie im Beispiel gezeigt. Würden Sie den `b`*-Tag im Quelltext oben z. B. vor dem* `i`*-Tag schließen, wäre das Dokument nicht mehr wohlgeformt.*

Veraltete Tags und Attribute

Einige Tags und Attribute sind mittlerweile durch neuere Features ersetzt worden. Dazu gehört z. B. der `font`-Tag, über den Sie die Schriftart eines Textes festlegen können: Statt `<font>` sollten Sie (bei diesem Beispiel) besser die Schriftart-Stile der Cascading Style Sheets (Kapitel 5) verwenden.

Veraltete Tags und Attribute werden von den gängigen Browsern wahrscheinlich noch einige Jahre unterstützt, weil in absehbarer Zeit wohl nur ein kleiner Teil der Websites im Internet auf die neuen Features umgestellt wird. Sie sollten in Ihren HTML-Dokumenten aber trotzdem immer die neuen Features einsetzen, damit Ihre Websites auch noch in zehn Jahren funktionieren. Im Buch habe ich bei veralteten Text und Attributen eine entsprechende Anmerkung angebracht.

Der Name und die ID von HTML-Elementen

Wenn Sie in JavaScript- oder ASP-Programmen auf HTML-Elemente zugreifen wollen (egal ob client- oder serverseitig), müssen Sie diese Elemente benennen. Und dazu stehen Ihnen gleich zwei Möglichkeiten zur Verfügung: Das `name`- und das `id`-Attribut. `name` ist eigentlich veraltet und in HTML 4.0 durch `id` ersetzt. Leider gehen die verschiedenen Browser ganz unterschiedlich mit diesen beiden Attributen um. Wenn Sie nur das `id`-Attribut zur Benennung verwenden, können Sie nicht davon ausgehen, dass Ihre HTML-Dokumente in allen Browsern funktionieren (was besonders dann gilt, wenn Sie intensiv mit DHTML und CSS arbeiten). Die umfangreichen Diskussionen über dieses Thema in den JavaScript-Newsgroups bestätigen die Verwirrung um diese Attribute. Befolgen Sie idealerweise den folgenden Hinweis, damit Sie sich nicht selbst verwirren (so wie ich bei der Recherche dieses Themas ...):

Benennen Sie ein HTML-Element grundsätzlich immer (oder wenigstens für die nächsten Jahre) mit `id` *und* `name` *(so wie es in vielen HTML-Quelltexten zu sehen ist). Die Wahrscheinlichkeit, dass Ihr Quelltext funktioniert (und das noch in verschiedenen Browsern!), wird damit nicht unwesentlich erhöht.*

Die gleichzeitige Verwendung von `id` und `name` ist kein Problem:

```
<img src="logo.gif" border="1" id="logo" name="logo">
```

Element-Arrays

Wenn Sie zwei HTML-Elementen den gleichen Namen geben, greifen Sie über einen nummerischen Index auf diese Elemente zu. Das erste Element in der Folge erhält automatisch den Index 0. In JavaScript verwenden Sie die Array-Syntax, um auf diese Elemente zuzugreifen. Der folgende Quellcode schreibt z. B. das aktuelle Datum und die Zeit in zwei gleich benannte Absätze:

```
<p id="date" name="date"></p>
<p id="date" name="date"></p>
<script language="JavaScript">
 currentDate = new Date;
```

```
 date[0].innerText = "Datum: " + currentDate.getDate() +
    "." + currentDate.getMonth() + "." +
    currentDate.getYear();
 date[1].innerText = "Zeit: " + currentDate.getHours() +
    ":" + currentDate.getMinutes() + ":" +
    currentDate.getSeconds();
</script>
```

4.1.5 Die Grundstruktur eines HTML-Dokuments

Eine HTML-Seite besitzt immer den folgenden Grundaufbau:

```
<html>
<head>
<title>Titel der Seite</title>
</head>
<body>
 ...
</body>
</html>
```


Am `html`-Tag erkennt der Browser, dass es sich um eine HTML-Datei handelt. Sie werden sich vielleicht wundern, dass dieser Tag notwendig ist, aber Browser können auch noch andere Dateien wie einfache Textdateien darstellen, die dann nicht nach dem HTML-Standard interpretiert werden. `<head>` definiert den Kopfbereich der Seite und enthält zunächst nur den `title`-Tag, der den Titel der Seite definiert. Dieser erscheint in der Titelleiste des Webbrowsers. Im `head`-Tag werden aber häufig auch Script-Programme untergebracht, um diese etwas vom eigentlichen Inhalt des Dokuments zu trennen. Den eigentlichen Inhalt finden Sie dann im `body`-Tag.

Der body-Tag

Der `body`-Tag definiert den Körper der HTML-Seite. Alles, was zwischen `<body>` und `</body>` steht, wird im Browser als das eigentliche Dokument angezeigt. In den Attributen dieses Tags können Sie die grundsätzliche Ansicht der Seite steuern. Diese Attribute sind aber mittlerweile durch Stile der Cascading Style Sheets ersetzt (Kapitel 5). Ein wichtiges Attribut ist `bgcolor`, das die Hintergrundfarbe des Dokuments definiert.

4.2 HTML-Dokumente gestalten

4.2.1 HTML-Kommentare

HTML-Kommentare sind spezielle Inhalte des Dokuments, die (wie üblich) nicht als Quelltext angesehen und folglich vom Browser ignoriert werden. Kommentare fügen Sie in die Zeichenketten `<!--` und `-->` ein:

```
<!--Das ist ein HTML-Kommentar -->
```

Mehrzeilige Kommentare sind kein Problem:

```
<!--Das ist ein mehrzeiliger
   HTML-Kommentar -->
```

4.2.2 Farben

Farben werden in HTML (wie in den meisten Programmiersprachen auch) in Form von 32-Bit-Integerwerten angegeben, wobei allerdings nur die ersten drei Byte verwendet werden. Der Wert des dritten Byte definiert den Rotanteil, der Wert des zweiten Byte den Grünanteil und der des ersten Byte den Blauanteil der Farbe. Ein solcher Farbwert wird deswegen auch als RGB-Wert bezeichnet.

Die meisten Programmierer verwenden die leicht lesbare hexadezimale Schreibweise zur Angabe von Farbwerten. Die hexadezimale Schreibweise beginnt in HTML mit einem »#« gefolgt von den Werten der einzelnen Bytes. Wie im hexadezimalen Zahlensystem üblich reichen die Werte der einzelnen Bytes von 00 (0) bis FF (255). Knallrot (meine Lieblingsfarbe) entspricht z. B. dem Wert #FF0000, Gelb (eine Mischung aus Rot und Grün) dem Wert #FFFF00. Wenn Sie sich ein wenig mit dem Mischen von Farben auskennen, werden Sie mit diesen Farbangaben keine Probleme haben (*ich* kenne mich damit nicht besonders gut aus und *habe* Probleme ...). Etwas einfacher ist die Verwendung von symbolischen Textkonstanten für die Grundfarben. Dabei wird einfach die englische Bezeichnung der Farbe verwendet. »red« steht z. B. für Rot. Die meisten Browser erkennen zumindest die 16 Basisfarben (deren Konstanten in HTML 4 standardisiert sind), moderne Browser erkennen aber auch wesentlich mehr dieser Konstanten.

HINWEIS

In SelfHTML finden Sie auf der Seite »HTML« | »Allgemeine Regeln für HTML« | »Farben definieren in HTML« im Abschnitt »Hexadezimale Angabe von Farben« ein Anzeigebeispiel für die 16 Grund- und die 265 Standardfarben. Unter »Speziellere Farbnamen« finden Sie auf derselben Seite die browserspezifischen Konstanten für die spezielleren Farben.

4.2.3 Grundlegendes zu Texten in HTML

Wenn Sie in einem HTML-Dokument Texte unterbringen, müssen Sie – im Vergleich zu einem normalen Text – einige wenige Eigenheiten beachten. Ältere Browser erkennen z. B. keine ASCII-Zeichen, deren Wert größer als 127 ist (dazu gehören z. B. die deutschen Umlaute). Neuere Browser können zwar mit allen ASCII-Zeichen umgehen, müssen aber denselben Zeichensatz verwenden wie das HTML-Dokument. Diese und die weiteren HTML-Text-Eigenheiten beschreibe ich in den folgenden Abschnitten.

Zeichensätze: Umlaute, das scharfe s und andere Sonderzeichen

Einige ältere Browser erlauben leider keine deutschen Umlaute, das scharfe s und andere Sonderzeichen im HTML-Quelltext, da sie nur den 7-Bit-ASCII-Zeichensatz unterstützen (die Umlaute und das scharfe s befinden sich im Bereich der ASCII-Zeichen 128-255, also in den Zeichen, die acht Bit verwenden). Umlaute, das scharfe s und andere Sonderzeichen können Sie für diese Browser mit besonderen Zeichenketten angeben (z. B. »ü« für »ü«). auf Seite listet die wichtigsten dieser Zeichenketten auf.

Neuere Browser arbeiten dagegen mit 8-Bit-ASCII-Zeichensätzen und können deshalb auch mit Umlauten und anderen Sonderzeichen umgehen. Da es im 8-Bit-Bereich aber viele verschiedene Zeichensätze gibt, hängt die Erkennung davon ab, ob der Browser denselben Zeichensatz verwendet wie das Dokument. In den meisten Browsern können Sie den Standard-Zeichensatz (der verwendet wird, wenn im HTML-Dokument kein spezieller Zeichensatz angegeben ist) in den Optionen einstellen. Im Internet Explorer finden Sie diese Option im Menü Extras | Optionen im Register Allgemein über den Schalter

Schriftarten, im Netscape über das Menü Ansicht | Zeichensatz. (Version 4.7) bzw. Anzeigen | Zeichenkodierung (Version 6.1)

Den zu verwendenden Zeichensatz können Sie aber auch im HTML-Dokument über einen Meta[1]-Tag angeben:

```
<meta http-equiv="Content-Type"
 content="text/html; charset=iso-8559-1">
```

In diesem Beispiel wird der in der westlichen Welt hauptsächlich verwendete ISO-Zeichensatz 8559-1 (benannt mit »Western Latin-1«) angegeben, der (natürlich) auch die deutschen Sonderzeichen enthält. Da dieser Meta-Tag Bestandteil des HTML 4.0-Standards ist, sollten alle neueren Browser damit umgehen können und folglich keine Probleme mit den deutschen Umlauten und dem scharfen s haben. Eine wichtige Voraussetzung dafür ist, dass Sie Ihre Dokumente in einem HTML-Editor erzeugen, der ebenfalls diesen Zeichensatz verwendet. Außerdem muss der Anwender Schriftarten im Browser eingestellt haben, die den Zeichensatz unterstützen. Normalerweise können Sie in westlichen Ländern davon ausgehen, dass das der Fall ist. Wenn Ihre Webseiten allerdings auch in anderen Ländern und mit allen Browsern funktionieren sollen, verzichten Sie lieber auf den Meta-Tag und verwenden die in beschriebenen speziellen HTML-Zeichenfolgen für Sonderzeichen.

In allen folgenden Beispielen (und in meinen kommerziellen ASP-Quellcodes) verwende ich die HTML-Darstellung der Umlaute und des scharfen s nicht, um den Quelltext lesbarer zu gestalten.

Die Zeichen »>«, »<«, »&« und »"« gehören ebenfalls zu den Sonderzeichen. Da diese Zeichen in HTML eine Sonderbedeutung besitzen, müssen sie grundsätzlich immer über die speziellen HTML-Zeichenfolgen in einen Text integriert werden.

1. Ein Meta-Tag enthält zusätzliche Informationen, die nicht Bestandteil des eigentlichen HTML-Dokuments sind. Viele HTML-Editoren verwenden Meta-Tags zur Speicherung programmspezifischer Informationen. Einige Meta-Tags werden allerdings auch vom Browser ausgewertet.

In Tabelle 4.1 finden Sie die HTML-Zeichenfolgen der wichtigsten Sonderzeichen.

Zeichen	HTML-Zeichenfolge
>	&gt;
<	&lt;
&	&amp;
"	&quot;

Tabelle 4.1: Die HTML-Zeichenfolgen für die wichtigsten Sonderzeichen

Neben diesen Zeichen gibt es noch eine große Anzahl weiterer Sonderzeichen, wie z. B. das griechische Alpha (α). Diese Zeichen sind in SelfHTML unter »Internationalisierung« | »Benannte Zeichen in HTML« beschrieben.

Leerzeichen und Zeilenumbrüche im Quelltext

Ein Webbrowser interpretiert Zeilenumbrüche und mehrfache Leerzeichen im Text normalerweise als einfache Leerzeichen. Aus:

```
Das ist ein
   Test.
```

wird:

```
Das ist ein Test.
```

Ist der Text länger, als der Browser in einer Zeile darstellen kann, bricht er den Text automatisch um.

Im Quelltext dienen Zeilenumbrüche und mehrfache Leerzeichen demnach lediglich dazu, diesen übersichtlicher zu gestalten. Wenn Sie definierte Zeilenumbrüche und mehrfache Leerzeichen jedoch auch im Ergebnis-Dokument benötigen, können Sie dazu eine spezielle Zeichenfolge und verschiedene Tags verwenden.

Mehrere Leerzeichen

Wenn Sie im Ergebnis mehrere Leerzeichen benötigen, können Sie das geschützte Leerzeichen verwenden. Verwenden Sie dazu die HTML-Zeichenfolge ` ` (non breaking space):

```
Hier folgen drei Leerzeichen:   .
```

Umbruchsteuerung

Der Browser bricht lange Texte normalerweise automatisch am rechten Rand um. Den automatischen Umbruch können Sie für einzelne Texte mit dem `nobr`-Tag verhindern. Der Benutzer muss dann nach rechts scrollen, um den Text weiterlesen zu können.

Sie können den Umbruch aber auch über HTML-Tags steuern. Der `br`-Tag (br = Break = (Um)Bruch) bewirkt einen einfachen Zeilenumbruch:

```
Das ist ein<br>
Test.
```

Dieser Tag wird einfach an das Ende der Zeile angehangen und nicht abgeschlossen.

Für wohlgeformte Dokumente ist vorgeschrieben, dass auch solche leeren Tags wie `<br>` *immer abgeschlossen werden müssen. Das funktioniert in der Kurzform auch, indem der Tag mit* `/>` *beendet wird:* `<br/>`*. Diese Regel ist allerdings nur bedingt auf HTML 4.0 anwendbar. Der Internet Explorer 5.x und Netscape 6.x haben zwar keine Probleme mit abgeschlossenen leeren Tags, Netscape 4.x kennt* `<br/>`*, ignoriert aber z. B.* `</hr>`*, andere Browser ignorieren abgeschlossene leere Tags komplett.*

Der `p`-Tag (p = Paragraph = Absatz) schließt einen Absatz ein:

```
<p>Das ist ein Absatz</p>
```

Ein Absatz besitzt etwas Abstand zu den vorhergehenden und den nachfolgenden Zeilen im Text. Viele Browser verlangen nicht, dass der `p`-Tag abgeschlossen wird. In einem wohlgeformten Dokument muss dieser Tag aber immer abgeschlossen werden.

Über das `align`-Attribut können Sie die Ausrichtung des Textes im Absatz bestimmen. Geben Sie dazu die Werte `left` (linksbündig), `center` (mittig), `right` (rechtbündig), `justify` (Blocksatz):

```
<p align= "justify">
  Das ist ein Absatz im Blocksatz.</p>
```

Wenn Sie den Text so ausgeben wollen, wie er eingegeben wird, können Sie den `pre`-Tag verwenden (pre = preformatted Text). Text, der in diesen Tag eingeschlossen wird, wird mit allen eingegebenen Zeilenumbrüchen und Leerzeichen ausgegeben.

```
<pre>
Das ist
präformatierter
Text. Dieser   Satz   enthält    viele    Leerzeichen.
</pre>
```

Die Schriftart ist allerdings die, die im Browser für »Nur Text« (Internet Explorer) bzw. »Schrift mit fester Breite« (Netscape) eingestellt ist (normalerweise ist dies Courier).

4.2.4 Formatierungen

Zur Formatierung des Textes stehen Ihnen einige Tags zur Verfügung. Der `b`-Tag z. B. formatiert den umschlossenen Text fett, der `h1`-Tag definiert eine Überschrift erster Ebene. Bei den meisten dieser Tags entscheidet normalerweise der Browser, wie die Formatierung im Ergebnis aussieht. Beachten Sie, dass Sie die Formatierung über Cascading Style Sheets (CSS) beeinflussen können. So können Sie z. B. den `h1`-Tag so umdefinieren, dass der Text immer in Arial, Schriftgröße 18 und in roter Farbe erscheint. Tabelle 4.2 listet die wichtigsten Formatiertags auf.

Tag	Bedeutung
`<b>`	fette Auszeichnung (bold)
`<i>`	kursive Auszeichnung (italic)

Tag	Bedeutung
<pre>	Mit dem Preformatted-Tag können Sie erreichen, dass Text so ausgegeben wird, wie er eingegeben wurde (mit allen Zeilenumbrüchen und Leerzeichen). Die Formatierung erfolgt allerdings in der Schriftart, die im Browser für »Nur Text« (Internet Explorer) bzw. »Schrift mit fester Breite« (Netscape) eingestellt ist (normalerweise ist dies Courier). Verwenden Sie diesen Tag für Quelltext-Listings und Ähnliches.
<h1> bis <h6>	Überschriften erster bis sechster Ebene (header). Überschriften werden in einem eigenen Absatz ausgegeben. Über das align-Attribut können Sie die Ausrichtung wie beim p-Tag bestimmen.

Tabelle 4.2: Die wichtigsten Formatiertags

Normaler Text wird vom Browser grundsätzlich in einer voreingestellten Schriftart formatiert. Um die Formatierung von normalem Text zu beeinflussen können Sie den veralteten font-Tag verwenden:

```
<font face="Times" size="4" color="#ff0000">
Das ist Text im veralteten &lt;font&gt;-Tag
</font>
```

Die Größe der Schrift wird mit einer Zahl zwischen 1 und 7 angegeben, wobei 1 etwa der Schriftgröße 7pt und 7 etwa der Schriftgröße 36pt entspricht.

Statt <font> verwenden Sie besser die Schriftart-Stile der Cascading Style Sheets (CSS). Diese bieten wesentlich mehr Möglichkeiten, sind HTML-konform und zukunftssicher. Da CSS in Kapitel 5 beschrieben wird, folgt hier nur ein einfaches (zum font-Beispiel äquivalentes) Beispiel. Zur Formatierung eines einfachen Textes wird der span-Tag verwendet (der nichts weiter bewirkt, als dass der umschlossene Inhalt als ein Element angesprochen werden kann):

```
<span style="color: red; font: 14pt Times">
 Das ist Text, dessen Formatierung mit CSS gestaltet wurde
</span>
```

4.2.5 Grafiken

Die meisten Browser können Dateien im GIF- und JPEG-Format darstellen. Diese Dateien können Sie über den `img`-Tag (img = Image = Abbildung) einbinden. Im `src`-Attribut (src = Source = Quelle) geben Sie die Quelldatei an. Befindet sich die Datei auf demselben Webserver, können Sie entweder eine relative Pfadangabe (ausgehend vom HTML-Dokument) oder eine vom Stammordner des Webservers ausgehende absolute Dateiangabe verwenden. Das `alt`-Attribut definiert einen alternativen Text für Browser, die das Bild aus irgendwelchen Gründen nicht anzeigen können (Tipp: Geben Sie hier immer "Pech gehabt" an, wer einen so alten Browser verwendet, ist selbst schuld).

Das folgende Beispiel bindet eine Bilddatei ein, die im selben Verzeichnis gespeichert ist, eine, die einen Ordner höher gespeichert ist, und eine in einem dem Stammordner untergeordneten Ordner:

```
<img scr="basket.gif" alt="Warenkorb">
<img scr="../shop.gif" alt="Shop">
<img scr="/images/remove.gif" alt="Löschen">
```

Sie können aber auch eine normale URL verwenden um ein Bild einzubinden, das auf einem anderen Webserver gespeichert ist:

```
<img scr="http://www.nitty-gritty.de/Media/header/header.gif">
```

Achten Sie dann aber darauf, dass Sie auch das Recht besitzen, die Datei zu verwenden.

4.2.6 Verweise

Innerhalb eines HTML-Dokuments können Sie auf Teile des Dokuments oder auf andere Dokumente bzw. auf beliebige Dateien mit einem HTML-Link verweisen (den Sie ja von vielen Websites her kennen). Dazu verwenden Sie den `a`-Tag (Anchor = Anker). Wollen Sie auf eine andere Datei verweisen, verwenden Sie die folgende Syntax:

```
<a href="Zielangabe">Verweistext</a>
<a href="Zielangabe"><img src="Grafikdatei"></a>
```

Das `href`-Attribut (Hyper(text) reference = Hypertext-Verweis) bezeichnet das Ziel des Verweises. Hier können Sie entweder eine Da-

teiangabe, eine URL oder einen Anker (siehe ab Seite 84) angeben. Als Verweis können Sie entweder einen Text oder (über den `img`-Tag) eine Grafik angeben.

Verweise auf lokale Dateien

Befindet sich die Datei auf demselben Webserver, kann diese mit einer normalen relativen Pfadangabe (ausgehend vom Standort des HTML-Dokuments) oder mit einer festen Pfadangabe (ausgehend vom Stammverzeichnis des Webservers) angegeben werden. Dabei verwenden Sie den Schrägstrich an Stelle des Backslashs.

Häufig wird auf Dateien im selben Verzeichnis verwiesen, dann geben Sie einfach keine Verzeichnisse an:

```
<a href="Products.htm">Unsere Produkte</a>
```

Befindet sich die andere Datei in einem Unterordner, geben Sie diesen einfach an:

```
<a href="Shop/Products.htm">Unsere Produkte</a>
```

Befindet sich die andere Datei z. B. in einem Ordner auf einer übergeordneten Ebene, können Sie mit zwei Punkten (die für den übergeordneten Ordner stehen) relativ nach oben verzweigen:

```
<a href="../Products.htm">Unsere Produkte</a>
```

Wollen Sie bei der Dateiangabe vom Stammordner des Webservers ausgehen, setzen Sie einen Schrägstrich vor die Dateiangabe:

```
<a href="/Shop/Products.htm">Unsere Produkte</a>
```

Verweise auf Dateien auf anderen Webservern

Eine Datei auf einem anderen Webserver können Sie mit einer URL adressieren:

```
<a href="www.nitty-gritty.de">Nitty-Gritty Homepage</a>
```

Wenn Sie lediglich die Webadresse des Servers angeben, adressieren Sie das Stammverzeichnis des Webservers. Wenn Sie keine Datei angeben, verwendet der Webserver das dort eingestellte Standarddokument (normalerweise DEFAULT.HTM). Natürlich können Sie wie im lokalen Web Ordner und Dateien angeben:

```
<a href="www.nitty-gritty.de/border/border.asp">
 Nitty-Gritty Homepage</a>
```

Verweise auf Anker in Dokumenten

In einem HTML-Dokument können Sie Anker anbringen, auf die Sie direkt verweisen können. Ein Verweis auf einen Anker führt den Anwender direkt zu dem so gekennzeichneten Textabschnitt. Ein Anker wird über den `a`-Tag angelegt, indem Sie im Attribut `name` einen Namen für den Anker angeben:

```
<a name="Ankername">HTML-Text</a>
```

Einen so definierten Anker können Sie im `href`-Attribut des `a`-Tags referenzieren, indem Sie ein # gefolgt vom Ankernamen angeben:

```
Verweis auf einen Anker in demselben Dokument
<a href="#Ankername">Verweistext</a>
Verweis auf einen Anker in einem anderen Dokument
<a href="Verweisziel#Ankername">Verweistext</a>
```

Verweise auf andere Dienste

Verweise können Sie nicht nur auf HTTP-Dienste erzeugen. Genauso können Sie über eine passende URL auf einen FTP- oder Maildienst verweisen. Eine E-Mail-Adresse geben Sie z. B. folgendermaßen an:

```
<a href="mailto:juergen.bayer@addison-wesley.de">Feedback zum Buch</a>
```

Klickt der Anwender auf diesen Verweis, öffnet sich normalerweise der auf dem Anwenderrechner installierte Mailclient.

Verweise, die eine neue Instanz des Browsers öffnen

Normalerweise wird die HTML-Seite, auf die ein Verweis zeigt, in derselben Instanz des Browsers geöffnet. Sie können den Verweis über das `target`-Attribut jedoch auch so einstellen, dass die neue Seite in einer neuen Instanz geöffnet wird:

```
<a href="Support.htm" target="_blank">Support</a>
```

4.2.7 Auflistungen und Aufzählungen

HTML definiert natürlich auch Tags für Auflistungen und Aufzählungen. Eine Auflistung wird dabei als »ungeordnete Liste« (unordered

List) bezeichnet, eine Aufzählung als »geordnete Liste« (ordered List). Verwenden Sie für eine Auflistung den `ul`-Tag:

```
<ul>
<li>Zaphod</li>
<li>Ford</li>
<li>Arthur</li>
<li>Trillian</li>
</ul>
```

Bild 4.1: Eine HTML-Auflistung

Wie Sie dem Beispiel entnehmen können, werden die einzelnen Listeneinträge über den `li`-Tag definiert.

Eine Aufzählung definieren Sie über den `ol`-Tag. Sie kann verschiedene Aufzählungszeichen besitzen, die Sie im (veralteten) `type`-Attribut definieren können. »1« definiert eine Aufzählung mit arabischen Ziffern, »A« bzw. »a« eine Aufzählung mit Buchstaben, »I« eine mit römischen Ziffern und »i« eine Aufzählung mit einem »i« als Aufzählungszeichen:

```
Was ich am liebsten mache:
<ol type="1">
<li>Motorradfahren</li>
<li>Carven</li>
<li>Snowboarden (demnächst)</li>
<li>Inliner laufen</li>
<li>Joggen</li>
<li>Feiern</li>
</ol>
```

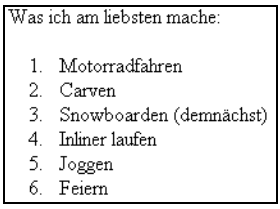


Bild 4.2: Eine HTML-Aufzählung mit Typ 1

Oh, sorry, ich habe bei der Auflistung doch glatt »Bücher schreiben« vergessen ...

4.2.8 Tabellen

HTML-Tabellen bieten hervorragende Möglichkeiten, Texte, aber auch Grafiken oder Eingabeelemente tabellenförmig anzuordnen. Viele HTML-Designer verwenden Tabellen nicht im eigentlichen Sinn (zur tabellenförmigen Darstellung von Texten), sondern um die HTML-Seite einfacher und besser gestalten zu können. Die Grundform einer Tabelle sieht folgendermaßen aus:

```
<table>           Beginn der Tabelle
<tr>              Beginn einer Zeile (Table Row)
<th>Inhalt</th>   Eine Überschrifts-Zelle (Table Header)
<td>Inhalt</td>   Eine normale Zelle (Table Data)
</tr>             Ende einer Zeile
<tr>              Beginn der nächsten Zeile
 ...
</tr>
</table>          Ende der Tabelle
```

Der `tr`-Tag leitet eine Zeile ein. In der Zeile werden mit `<th>` oder `<td>` einzelne Zellen definiert. Jede Zeile sollte dieselbe Anzahl Zellen besitzen wie alle anderen Zeilen in der Tabelle, damit die Tabelle korrekt dargestellt wird. Ausnahmen bilden Zellen, die über mehrere Spalten oder Zeilen definiert sind (Seite 89).

Eine Überschrifts-Zelle (`<th>`) wird normalerweise einfach mit fett und zentriert formatiertem Inhalt dargestellt, eine normale Zelle wird normal und linksbündig formatiert.

Im Attribut `border` des `table`-Tag können Sie die Breite des Tabellenrahmens in Pixel angeben. Wenn Sie hier o eintragen, ist der Rahmen der Tabelle unsichtbar. Tabellen mit unsichtbaren Rahmen werden häufig verwendet, um Darstellungsprobleme, die mit HTML-Elementen sonst entstehen, zu lösen. Mit diesen Tabellen können Sie HTML-Elemente sehr schön in einzelnen Spalten untereinander anordnen (siehe Seite 87).

Über das `table`-Tag-Attribut `cellpadding` können Sie angeben, wie viele Pixel der Text in der Zelle vom Tabellenrand entfernt ist. Dieses Attribut ist für Tabellen mit sichtbarem Rahmen recht wichtig, um ein wenig Abstand zwischen Rahmen und Text zu erzeugen. Das Attribut `cellspacing` bestimmt den Abstand der einzelnen Zellen voneinander und vom Tabellenrahmen. Das folgende Listing zeigt die Definition einer einfachen Tabelle mit einem 1-Pixel breiten Rahmen und einem inneren Rand in jeder Zelle von 5 Pixeln:

```
<table border="1" cellpadding="5">
   <tr>
   <th>Vorname</th>
   <th>Nachname</th>
   </tr>
   <tr>
   <td>Zaphod</td>
   <td>Beeblebrox</td>
   </tr>
   <tr>
   <td>Philo</td>
   <td>Dufresne</td>
   </tr>
   </table>
```

Abbildung 4.3 zeigt das Ergebnis.

Vorname	Nachname
Zaphod	Beeblebrox
Philo	Dufresne

Bild 4.3: Eine einfache HTML-Tabelle

Unsichtbare Tabellen und Eingabeelemente

In den Zellen können Sie, wie bereits erwähnt, auch Eingabeelemente und Grafiken unterbringen. Tabellen eignen sich damit hervorragend zur Ausrichtung von HTML-Elementen, die ansonsten schwierig zu positionieren sind. Das folgende Listing verwendet eine Tabelle dazu, Eingabeelemente und erklärenden Text bündig untereinander anzuordnen:

```
<table border="0">
<tr>
<td>Vorname: </td>
<td><input type="Text" name="Vorname"></td>
</tr>
<tr>
<td>Nachname: </td>
<td><input type="Text" name="Nachname"></td>
</tr>
<tr>
<td>Ort: </td>
<td><input type="Text" name="Ort"></td>
</tr>
</table>
```

Das Ergebnis zeigt Abbildung 4.4.

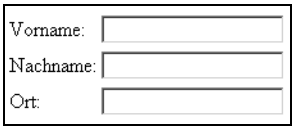


Bild 4.4: Eine rahmenlose HTML- Tabelle mit Eingabeelementen

Beachten Sie, dass Sie HTML-Elemente mit den Positionierungs-Stilen der Cascading Style Sheets auch frei auf dem HTML-Dokument positionieren und damit auf Tabellen verzichten können.

Die Breite und Höhe der Tabelle und der Spalten

Wenn Sie für die Tabelle bzw. die Zellen/Spalten keine Breite angeben, richtet sich die Breite der Spalten und damit der Tabelle nach dem Inhalt der Zellen. Dasselbe gilt für die Höhe. Die Breite und die Höhe können Sie jedoch auch für die ganze Tabelle oder für einzelne Zellen über die Attribute `width` und `height` bestimmen. Hier können Sie Werte in Pixel (ohne Einheit) oder in Prozent (mit Prozentzeichen) verwenden. Eine Tabelle, die immer 80 Prozent der Breite des Browserfensters einnimmt, wird so definiert:

```
<table width="80%">
```

Wenn Sie die Breite einzelner Spalten definieren wollen, könnten Sie dem `th`- bzw. dem `td`-Tag das `width`-Attribut zufügen. Die Breite der Spalte richtet sich dann nach der breitesten Zelle. Es reicht dann aus, die Breite der Zellen in der ersten Zeile zu definieren. Alternativ können Sie den optionalen Tag `<colgroup>` verwenden, um die Breite aller Spalten strukturierter einzustellen:

```
<table border="1">
<colgroup>
<col width="120">
<col width="150">
</colgroup>
<tr>
<th>Vorname</th>
<th>Nachname</th>
</tr>
<tr>
<td>Zaphod</td>
<td>Beeblebrox</td>
</tr>
<tr>
<td>Philo</td>
<td>Dufresne</td>
</tr>
</table>
```

Mehrspaltige Zellen

Sie können einzelne Zellen auch über mehrere Spalten definieren, indem Sie einfach das `colspan`-Attribut verwenden:

```
<table border="1">
<tr>
<td>Einfache Zelle</td>
<td>Einfache Zelle</td>
<td>Einfache Zelle</td>
</tr>
<tr>
<td>Einfache Zelle</td>
<td colspan="2">Zelle über zwei Spalten </td>
```

```
</tr>
</table>
```

Einfache Zelle	Einfache Zelle	Einfache Zelle
Einfache Zelle	Zelle über zwei Spalten	

Bild 4.5: Eine HTML-Tabelle mit einer Zelle, die über zwei Spalten definiert ist

Mehrzeilige Zellen

Genau wie bei mehrspaltigen Zellen können Sie Zellen auch über mehrere Zeilen definieren. Verwenden Sie dazu das Argument `rowspan`.

Ausrichtung des Textes

Für jede Zelle können Sie die Ausrichtung des Textes über die Attribute `aling` (horizontale Ausrichtung) und `valing` (vertikale Ausrichtung) festlegen. Die möglichen Werte für `aling` sind `left`, `center` und `right`, die Werte für `valign` sind `top`, `middle` und `bottom`.

Tipps und Tricks

Leere Zellen korrekt darstellen

Eine Tabelle enthält oft leere Zellen, besonders dann, wenn die Tabelle dynamisch mit ASP erzeugt wird (z. B. aus den Daten einer Datenbanktabelle). Wenn Sie in eine leere Zelle einfach nichts oder ein Leerzeichen schreiben, zeigt der Browser die Zelle in der Regel einfach nicht an. Besonders unschön ist dies, wenn die Tabelle einen Rahmen besitzt:

```
<table border="1">
<tr>
<th>Leere Zelle</th>
<th>Volle Zelle</th>
</tr>
<tr>
<td></td>
<td>Test</td>
</tr>
</table>
```

Leere Zelle	Volle Zelle
	Test

Bild 4.6: Eine HTML-Tabelle mit einer leeren Zelle ohne den »Trick«

Wenn Sie für leere Zellen stattdessen das geschützte Leerzeichen verwenden (` `), wird die Zelle korrekt dargestellt, wie Abbildung 4.7 zeigt.

```
<table border="1">
<tr>
<th>Leere Zelle</th>
<th>Volle Zelle</th>
</tr>
<tr>
<td> </td>
<td>Test</td>
</tr>
</table>
```

Leere Zelle	Volle Zelle
	Test

Bild 4.7: Eine HTML-Tabelle mit korrekt dargestellter leerer Zelle

4.2.9 Layer

Über die Tags `<div>` und `<span>` können Sie HTML-Elemente und Text gruppieren. So können Sie mehrere HTML-Elemente gleichzeitig oder reinen Text beeinflussen. Der folgende Quelltext sorgt z. B. für eine zentrierte Ausrichtung eines Textes:

```
<div align="center">
 Das ist ein einfacher Bereich zur Änderung der
 Ausrichtung von einfachem Text
</div>
```

Mit DHTML und CSS können Sie Layer ebenfalls sehr gut einsetzen. So können Sie einen Layer z. B. auf eine andere Position setzen, den Layer sichtbar und unsichtbar schalten (z. B. um Menüs auf- und zuzuklappen) und den Inhalt des Layers dynamisch verändern. Mit CSS

können Sie den Stil eines Layers flexibel anpassen. So können Sie den Stil normalen Textes ändern, was in HTML nur bedingt (über den `font`-Tag) möglich ist. Ein einfaches Beispiel:

```
<div style="border: thin solid black;
 background-color:silver; color:red; font: Verdana 18pt">
 Das ist ein einfacher div zur Änderung der
 Formatierung von einfachem Text</div>
```

`div` und `span` unterscheiden sich dadurch, dass der Inhalt eines `div`-Tag immer in einem Absatz ausgegeben wird, bei `span` ist das nicht der Fall:

```
<div style="border: thin solid black;
 background-color:silver; color:red; font: Verdana 18pt">
 Das ist ein einfacher Bereich (mit div) zur Änderung der
 Formatierung von einfachem Text
 </div>
 Text nach dem div.

<span style="border: thin solid black;
 background-color:silver; color:red; font: Verdana 18pt">
 Das ist ein einfacher Bereich (mit span) zur Änderung der
 Formatierung von einfachem Text
</span>
Text nach dem span.
```

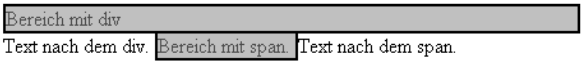


Bild 4.8: HTML-Bereiche mit div und span

4.2.10 Formulare

HTML-Formulare werden normalerweise dazu verwendet, den Inhalt der in einem Formular enthaltenen Eingabeelemente an eine E-Mail-Adresse, ein ASP-Dokument oder eine anderes Internetprogramm, wie z. B. eine CGI-Anwendung, zu senden. Das Formular wird dazu über einen speziellen Submit-Schalter oder über spezielle Techniken (die im Online-Artikel »ASP-Tipps und Tricks« beschrieben werden) vom Anwender bestätigt.

Das Internetprogramm wertet die Eingaben aus und regiert darauf. Wie das programmiert wird, zeige ich in Kapitel 9 noch sehr ausführlich.

Ein Formular wird immer in den `form`-Tag eingeschlossen. Darin definieren Sie normale HTML-Elemente, aber natürlich auch Eingabeelemente. Der `form`-Tag besitzt einige wichtige Attribute:

Attribut	Bedeutung
`name`	In diesem Attribut geben Sie den Namen des Formulars an. Der Name wird hauptsächlich in clientseitigen JavaScript-Programmen zur Referenzierung des Formulars verwendet.
`method`	`method` definiert, wie die im Formular eingegebenen Daten gesendet werden. Wenn Sie hier den Wert `post` angeben (= übertragen), werden die Daten direkt an die Standardeingabe der angegebenen Adresse gesendet. In ASP lesen Sie die einzelnen eingegebenen Daten über `Request.Form` aus. Verwenden Sie die Methode `get`, werden die Daten in die Server-Umgebungsvariable `QUERY_STRING` übertragen. In ASP verwenden Sie dann `Request.QueryString`, um die einzelnen Eingaben auszulesen.
`action`	`action` definiert, welche Aktion ausgeführt werden soll, wenn der Anwender das Formular bestätigt. Wenn Sie eine E-Mail-Adresse angeben (mailto:*Name@Server*), werden die Formulareingaben als E-Mail an die angegebene Adresse versendet (vorausgesetzt der E-Mail-Versand ist auf dem System möglich). Wenn Sie bei der Bestätigung des Formulars ein Programm aufrufen wollen (das die Eingaben auswertet), geben Sie im `action`-Argument den Dateinamen als Dateipfad oder URL an. Ein Dateipfad kann (wie immer ...) relativ (ausgehend vom Standort des Dokuments, das das Formular enthält) oder absolut (ausgehend vom Stammordner des Webservers) angegeben werden.

Tabelle 4.3: Die Attribute des form-Tag

Ein Formular wird normalerweise mit einem Submit- und einem Reset-Schalter ausgerüstet. Ein Klick auf den Submit-Schalter versendet das Formular, ein Klick auf den Reset-Schalter setzt die Eingaben auf die Voreinstellungen zurück. Ein Formular, das an eine ASP-Seite gesendet werden soll, wird so definiert:

```
<form name="Register" method="post"
 action="EvaluateForm.asp">
```

```
<-- Hier werden die Eingabeelemente definiert.
    Das Beispiel verwendet zwei Textfelder
    für die Eingaben -->
<table border="0">
  <tr>
   <td>Vorname:</td>
   <td><input type="text" name="Vorname"></td>
  </tr>
  <tr>
   <td>Nachname:</td>
   <td><input type="text" name="Nachname"></td>
  </tr>
</table>
<-- Der Submit- und der Reset-Schalter --> <p>
<input type="submit" value="Absenden"> 
<input type="reset" value="Zurücksetzen">
</p>
</form>
```

Das Beispiel setzt bereits eine Tabelle ein, um die Beschriftungen und die Eingabeelemente untereinander ausrichten zu können.

Das Formular ist im Ergebnis per Default nicht sichtbar (mit Style-Angaben können Sie das Aussehen allerdings beeinflussen). Abbildung 4.9 zeigt die Ausgabe im Internet Explorer.

Bild 4.9: Ein HTML-Formular im Internet Explorer

Bild als Submit-Schalter

Über den `imput`-Tag können Sie auch ein Bild als Submit-Schalter festlegen:

```
<form method="post" action="evaluate.asp">
 ...
 <input type="image" name="ok" id="ok" scr="ok.gif">
</form>
```

Klickt der Anwender auf das Bild, wird das Formular damit bestätigt.

4.2.11 Steuerelemente (Eingabeelemente)

HTML 4.0 definiert einige Eingabeelemente, wie z. B. ein Textfeld oder ein Listenfeld. Diese Elemente werden meist dazu verwendet, den Benutzer etwas eingeben zu lassen und diese Eingaben serverseitig über ASP oder ähnliche Technologien auszuwerten. Da die HTML-Eingabenelemente den Standard-Windows-Steuerelementen sehr ähnlich sind, verzichte ich auf eine nähere Beschreibung der Funktionsweise. Ich gehe davon aus, dass Sie die Windows-Steuerelemente bereits (von der Anwendung her) kennen.

Beachten Sie, dass manche ältere Browser (wie der Netscape 4.7) Steuerelemente nur dann auf der Seite darstellen, wenn diese in ein Formular eingebettet sind. Sie sollten also immer einen `form`*-Tag um die Steuerelemente auf Ihren Seiten legen, auch wenn Sie die Funktionalität eines Formulars gar nicht benötigen (für diesen Tipp habe ich mich etwa eine halbe Stunde über den Netscape 4.7 geärgert, der auf einer Seite ohne* `form`*-Tag meine Steuerelemente einfach nicht angezeigt hat).*

Die Tabulatorreihenfolge

Die Tabulatorreihenfolge ist per Voreinstellung so eingestellt, dass der Eingabecursor vom ersten Element der Seite automatisch zum nächsten springt. Wenn Sie die Tabulatorreihenfolge ändern wollen, können Sie dies über das Attribut `tabindex` erledigen, dem Sie eine Zahl zwischen 0 und 32767 zuweisen. Die Navigation erfolgt dann vom Element mit dem niedrigsten Index zu dem Element mit dem höchsten.

Schreibgeschützte und deaktivierte Steuerelemente

Über das boolsche Attribut `disabled` können Sie erreichen, dass ein Steuerelement deaktiviert ist. Solche Steuerelemente werden oft verwendet, um dem Anwender etwas anzuzeigen, den Wert aber nicht editieren zu lassen. Der Inhalt des Steuerelements wird in einer etwas schwächeren Farbe dargestellt und der Anwender kann den Eingabecursor nicht in das Element setzen. Das ebenfalls boolsche Attribut `readonly` bewirkt Ähnliches, nur dass der Inhalt normal dargestellt wird und der Anwender den Eingabecursor in das Element setzen kann, um den Wert z. B. in die Zwischenablage zu kopieren.

`readonly` *und* `disabled` *werden nicht von allen Browsern unterstützt. Der Netscape 4.7 kann z. B. damit nichts anfangen und stellt das Steuerelement ganz normal dar. Unterstützt werden diese Attribute vom Internet Explorer ab Version 4.x und vom Netscape ab Version 6.x.*

Einzeilige Textfelder

Einzeilige Textfelder, in die der Anwender einen Text eingeben kann, werden über den `input`-Tag eingerichtet. Schreiben Sie in das Attribut `type` dazu den Wert »text«. Im Attribut `size` können Sie die Breite des Elements in Zeichen bestimmen. Das Attribut `maxlength` bestimmt die maximal eingebbare Textlänge. Das folgende Beispiel erzeugt ein 20 Zeichen großes Textfeld mit maximal 50 Zeichen Eingabemöglichkeit:

```
<input type="text" name="city" id="city" size="20"
 maxlength="50">
```

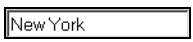


Bild 4.10: Ein einzeiliges HTML-Textfeld

Lassen Sie das Attribut `maxlength` weg, ist die Eingabe theoretisch unbegrenzt (aber wohl immer durch das Betriebssystem eingeschränkt).

Im Attribut `value` können Sie einen Text voreinstellen:

```
<input type="text" name="city" id="city" size="20"
 maxlength="50" value="New York">
```

Aus diesem Attribut lesen Sie später den eingegebenen Wert auch wieder aus.

Passwort-Textfelder

Ein Textfeld, das die Eingabe nicht als Klartext anzeigt, sondern für die einzelnen Zeichen Sterne anzeigt, erzeugen Sie, indem Sie ein `input`-Element mit `type="password"` erzeugen. Die Attribute entsprechen denen des einzeiligen Textfelds:

```
<input type="password" name="password" id="password"
 size="20" maxlength="50">
```

Bild 4.11: Ein HTML-Textfeld für Passworteingabe

Beachten Sie, dass in einem solchen Textfeld nur die Ansicht der Eingabe verschlüsselt ist. Die Übertragung der Daten vom Browser zum Server erfolgt trotzdem im Klartext.

Dateiauswahl-Textfelder

Ein `input`-Element vom Typ `file` ermöglicht die Auswahl eines Dateinamens über einen automatisch neben dem Element angelegten Schalter:

```
<input type="file" name="file" id="file" size="50">
```

Bild 4.12: Ein HTML-Textfeld für eine Dateiauswahl

Betätigt der Anwender den Schalter, zeigt der Browser einen Standard-Dateiauswahl-Dialog zur Auswahl der Datei an. Solche Steuerelemente werden üblicherweise dazu verwendet, eine Datei auf den Server hochzuladen. Das Hochladen von Dateien wird im Online-Artikel »ASP-Tipps und Tricks« beschrieben.

Mehrzeilige Textfelder

Textfelder, in die der Anwender mehrere Zeilen eingeben kann, erzeugen Sie nicht über den `input`-Tag, sondern über den `textarea`-Tag. Über die Attribute `rows` und `cols` steuern Sie die Anzahl der sichtbaren Zeilen bzw. Zeichen (Spalten):

```
<textarea name="message" id="message" rows="5" cols="30">
</textarea>
```

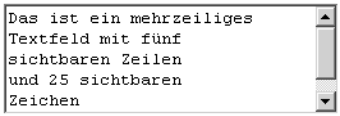


Bild 4.13: Ein mehrzeiliges HTML-Textfeld

Die Anzahl der sichtbaren Zeilen und Spalten beeinflusst nicht die Eingabemöglichkeit. Der Benutzer kann auch wesentlich mehr Zeilen und Zeichen pro Zeile eingeben, als sichtbar sind. Zeilen werden allerdings in den meisten Browsern (Internet Explorer ab Version 5.x, Netscape ab Version 6.x) per Voreinstellung automatisch am rechten Rand umbrochen. Den Umbruch können Sie über das Attribut `wrap` steuern (das nicht von allen Browsern, aber vom Internet Explorer und vom Netscape unterstützt wird):

- `wrap="off"`: Der Browser bricht lange Zeilen nicht um. Voreinstellung für den Netscape 4.x.
- `wrap="virtual"`: Der Browser bricht lange Zeilen um. Der Umbruch erfolgt aber so, dass beim Auswerten die Zeilen so zurückgegeben werden, wie sie eingegeben wurden. Voreinstellung für den Netscape 6.x und den Internet Explorer 5.x.
- `wrap="physical"`: Der Browser bricht lange Zeilen um. Bei der Auswertung stehen an den automatischen Umbruchstellen echte Zeilenumbrüche.

Das Textfeld wird automatisch mit Rollbalken versehen, sofern dies notwendig ist.

Wenn Sie das Textfeld mit einem Text vorbelegen wollen, tragen Sie den Text mit allen Leerzeichen und Zeilenumbrüchen (anders als in normalem HTML) einfach in den `textarea`-Tag ein:

```
<textarea name="message" id="message" rows="5" cols="30">
Das ist ein mehrzeiliges
Textfeld mit fünf
sichtbaren Zeilen
und 25 sichtbaren
Zeichen
</textarea>
```

Beachten Sie, dass der hier eingetragene Text nicht als HTML-Quelltext ausgewertet wird. Sie können also auch HTML-Sonderzeichen verwenden ohne diese maskieren zu müssen. Das beinhaltet eben auch, dass Leerzeichen so ausgegeben werden, wie diese im Quelltext stehen. Um im Ergebnis keine Leerzeichen vor den einzelnen Zeilen stehen zu haben, muss der Inhalt des `textarea`-Tags immer linksbündig gespeichert werden, auch dann, wenn der Tag selbst eingerückt ist.

Listenfelder

Listenfelder bieten dem Anwender eine Auswahl aus einer Liste von Werten an. Zur Erzeugung eines Listenfelds verwenden Sie den `select`-Tag. Die einzelnen Werte geben Sie in `option`-Tags an. Eine einfache Liste, aus der der Anwender einen einzelnen Wert auswählen kann, legen Sie z. B. so an:

```
<select name="car" id="car" size="3">
  <option>Ford Puma</option>
  <option>Fiat Panda</option>
  <option>Porsche 911</option>
  <option>Audi TT</option>
</select>
```

Das `size`-Attribut legt fest, wie viele Zeilen in der Liste sichtbar sind. Enthält die Liste mehr Zeilen, wird diese automatisch mit einem Rollbalken versehen.

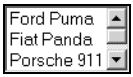


Bild 4.14: Ein einfaches HTML-Listenfeld

Über das boolsche `selected`-Attribut des `option`-Tag können Sie ein Element der Liste vorauswählen:

```
...
  <option selected="selected">Fiat Panda</option>
...
```

Wenn Sie das `size`-Attribut weglassen oder auf den Wert 1 setzen, wird das Listenfeld einzeilig mit einem Pfeil am rechten Rand dargestellt:

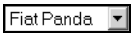


Bild 4.15: Ein einzeiliges HTML-Listenfeld

Mehrzeilige Listenfelder können Sie über das boolsche Attribut `multiple` so einstellen, dass diese eine Mehrfachauswahl zulassen:

```
...
<select name="car" id="car" multiple="multiple" size="3">
...
```

Den Anwender kann (wie in Windows üblich) über eine Kombination von Maustaste und Strg bzw. ⇧ mehrere Elemente auswählen.

Bild 4.16: Ein Mehrfachauswahl-Listenfeld in HTML

In einem Mehrfachauswahl-Listenfeld können Sie natürlich mehrere Listenelemente über das `selected`-Attribut des `option`-Tag vorauswählen.

Tipps und Tricks

Die Breite von Listenfeldern definieren

Listenfelder werden vom Browser normalerweise so breit angelegt, dass der Text der Einträge gut sichtbar ist. Über einen CSS-Stil (für den IE) und das `width`-Attribut (für den Netscape) können Sie die Breite aber auch festlegen (`px` steht hier für Pixel):

```
<select name="car" id="car" size="3" style="width=100px"
 width="100px">
```

Optionbuttons

Optionbuttons ermöglichen die Auswahl aus einer von mehreren vorgegebenen Optionen. Diese Elemente werden über den `input`-Tag mit `style="radio"` angelegt:

```
Ihr Geschlecht:<br>
<input type="radio" name="sex" id="sex"
 value="woman">Weiblich<br>
<input type="radio" name="sex" id="sex"
 value="man" checked="checked">Männlich<br>
<input type="radio" name="sex" id="sex"
 value="unknown">Weiß nicht
```

Über das boolsche Attribut `checked` können Sie einen der Schalter voreinstellen.

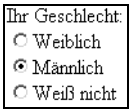


Bild 4.17: HTML-Optionbuttons

Alle Optionbuttons mit demselben Namen gehören derselben Gruppe an. Aus einer Gruppe kann immer nur ein Schalter eingeschaltet werden. Wenn Sie z. B. zwei verschiedene Gruppen benötigen, müssen Sie auch zwei verschiedene Namen verwenden:

```
Ihr Geschlecht:<br>
<input type=radio name="sex" value="woman">Weiblich<br>
<input type="radio" name="sex" value="man"
 checked="checked">Männlich<br>
<input type=radio name="sex" value="unknown">
Weiß nicht<br>
 <br>
Ihr bevorzugtes Fahrzeug:<br>
<input type=radio name="vehicle" value="woman">Fahrrad<br>
<input type="radio" name="vehicle" value="man" checked="chek-
ked">Motorrad<br>
<input type=radio name="vehicle" value="unknown">Auto
```

Checkboxen

Checkboxen ermöglichen ähnlich Optionbuttons die Auswahl von Optionen. Checkboxen lassen aber immer auch die mehrfache Auswahl zu und können wieder ausgeschaltet werden. Checkboxen legen Sie ähnlich wie Optionbuttons über den `input`-Tag mit `type="checkbox"` an. Sie können auch hier Schalter voreinstellen, nur dass natürlich auch mehrere möglich sind:

```
Ihre Hobbies:<br>
<input type="checkbox" name="hobbies"
 value="motorcycle" checked="checked">Motorrad fahren<br>
<input type="checkbox" name="hobbies"
 value="snowdoarden" checked="checked">Snowboarden<br>
<input type="checkbox" name="hobbies"
 value="inlinern" checked="checked">Inlinern
```

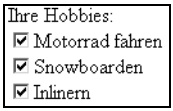


Bild 4.18: HTML-Checkboxen

Schalter

HTML kennt prinzipiell drei verschiedene Arten von Schaltern: Ein Submit-Schalter wird in einem Formular verwendet, um die eingegebenen Daten abzusenden, die Betätigung eines Reset-Schalters führt dazu, dass die Eingaben eines Formulars zurückgesetzt werden (vgl. Seite 92). Die dritte Art, ein normaler Schalter, wird normalerweise mit einem JavaScript-Programm verknüpft, das bei der Betätigung des Schalters aufgerufen wird.

Schalter werden über den `input`-Tag eingerichtet. Das `type`-Attribut legt den Typ des Schalters fest, im `value`-Attribut geben Sie die Beschriftung des Schalters an:

```
<!-- Submit-Schalter -->
<input type="submit" value="Absenden">
<!-- Reset-Schalter -->
<input type="reset" value="Zurücksetzen">
<!-- normaler Schalter -->
<input type="button" value="Hilfe" onclick="showHelp();">
```

Wie Sie dem Beispiel entnehmen können, wird ein normaler Schalter über das `onclick`-Attribut mit einer JavaScript-Funktion verknüpft. Um den Quellcode komplett zu machen, folgt hier nur kurz die JavaScript-Funktion, die für das Beispiel eine Hilfeseite öffnet:

```
<script language="JavaScript">
function showHelp()
{
   window.open("help.htm");
}
</script>
```

Tipps und Tricks

Die Breite von Schaltern definieren

Schalter werden vom Browser in der Regel automatisch so breit angelegt, dass die Beschriftung gut auf den Schalter passt. Das führt eigentlich immer dazu, dass mehrere Schalter auf einer HTML-Seite unterschiedlich breit sind. Über einen CSS-Stil (für den Internet Explorer) und das `width`-Attribut (für den Netscape) können Sie die Breite jedoch auch festlegen:

```
<input type="button" value="Hilfe" style="width:8em"
 width="8em" onclick="showHelp();">
```

Die in diesem Beispiel verwendete Einheit `em` steht für die Höhe eines Zeichens in der verwendeten Schriftart. 8em bedeutet also, dass der Schalter achtmal so breit ist wie ein Zeichen hoch. Die möglichen Einheiten für Breiten- und Längenangaben werden in Kapitel 5 bei den Cascading Style Sheets beschrieben.

Versteckte Felder

Wenn Sie mit ASP oder anderen serverseitigen Technologien programmieren, werden Sie recht bald den Bedarf haben, Daten auf einer Seite zu speichern, die der Anwender jedoch nicht sehen darf. Häufig müssen diese Daten sogar noch an das auswertende Programm übermittelt werden, wenn der Anwender ein Formular bestätigt. Genau dafür werden versteckte Felder verwendet. Ein solches Feld ähnelt einem Textfeld, nur dass dieses eben nicht sichtbar ist:

```
<input type="hidden" name="recipient" id="recipient"
 value="donald@duck.ent">
```

Auf versteckte Felder können Sie in JavaScript und in ASP zugreifen wie auf jedes andere Feld. Den gespeicherten Wert lesen Sie aus der Eigenschaft `value` aus.

Kapitel 9 geht noch einmal auf versteckte Felder in ASP ein.

Beachten Sie, dass versteckte Felder zwar nicht im Browser sichtbar sind, aber schon aus dem Quelltext ausgelesen werden können. Vertrauliche Daten sollten also nicht in solchen Feldern gespeichert werden.

5 Cascading Style Sheets

Eine Website ohne Cascading Style Sheets ist langweilig (außer, wenn die Website mit Flash animiert ist ☺). Erst Cascading Style Sheets ermöglichen eine optisch ansprechende und interessante Gestaltung der HTML-Dokumente. In Zusammenarbeit mit DHTML erlaubt CSS wichtige Features wie z. B. das dynamische Ändern von Inhalten bei Mausbewegungen auf Elementen und Menüs. An CSS kommen Sie also nicht vorbei (außer, Sie beabsichtigen, eine langweilige Website zu gestalten ... oder Flash zu verwenden). Dieses Kapitel zeigt die Grundlagen von CSS, kann aber nicht auf alle Features eingehen. CSS ist eben (wie so vieles ...) recht komplex. Schauen Sie einfach in die offizielle CSS-Dokumentationen, wenn Sie mehr wissen wollen. Sie finden diese Dokumentationen auf der Seite *http://www.w3.org/TR/REC-CSS1* und *http://www.w3.org/TR/REC-CSS2*.

5.1 Was sind Cascading Style Sheets?

Eins vorweg: Der Begriff »Cascading Style Sheets« (CSS) führt häufig zu einer leichten Verwirrung. Es handelt sich nämlich zunächst einmal um (im Vergleich zu Standard-HTML) erweiterte Stile für HTML-Elemente. Dass diese auch kaskadierend aufeinander aufbauen können, ist für das anfängliche Verständnis gar nicht wichtig.

CSS ist eine Sprache, mit der Sie den Stil von HTML-Elementen wesentlich flexibler beeinflussen können, als mit HTML alleine. Dazu gehören z. B. die Hintergrundfarbe, die Rahmenart, die Schriftart, die Position, die äußeren Ränder, die innere Einrückung und die Sichtbarkeit von HTML-Elementen. Einen Absatz können Sie z. B. mit einem Rahmen und einer Hintergrundfarbe definieren:

```
<p style="font: bold 14pt Times;
  border: thin solid black;
  background-color: silver;
  padding: 5 5">
Das ist ein Absatz<br>
mit CSS-Stilen
</p>
```

Bild 5.1: Ein HTML-Absatz mit CSS-Stilen

Mit CSS ist zunächst einmal wesentlich mehr Formatierung möglich als mit HTML alleine. Optisch gut gestaltete Websites kommen deshalb heutzutage ohne CSS gar nicht mehr aus. Features wie die dynamische Positionierung und das Verbergen und Wieder-Anzeigen von Elementen ermöglichen zudem (in Verbindung mit DHTML) die Erzeugung benutzerfreundlicher und cooler Websites mit Menüs und anderen dynamischen Elementen.

CSS ermöglicht aber nicht nur die Anpassung des Stils einzelner Elemente. Sie können auch die Stile ganzer Gruppen von Elementen definieren. Eine im Internet übliche Technik ist, die Stile der verwendeten HTML-Tags mit CSS zu definieren und in einer separaten Datei zu speichern. Diese Stil-Datei wird dann über einen HTML-Link in jede HTML-Datei der Website eingebunden. So können Sie den Stil einer ganzen Website sehr einfach ändern, indem Sie einfach die Stil-Datei anpassen.

CSS 1 ist in HTML 4 standardisiert, moderne Browser sollten also mit den enthaltenen Stilen einigermaßen zurecht kommen. Mittlerweile existiert auch ein neuer Standard, die Version 2. Diese Version wird von den einzelnen Browsern nur sehr mäßig unterstützt.

HiN/WEIS

Auf der Seite http://www.webreview.com/style/css1/charts/mastergrid.shtml *finden Sie eine Übersicht über die Unterstützung der einzelnen CSS-1-Features durch die einzelnen Browser. Die Seite* http://www.webreview.com/style/css2/charts/selectors. shtml *leistet Ähnliches für CSS 2.*

Natürlich hat Microsoft (und wahrscheinlich auch Netscape) den Standard wieder einmal durch eigene Stile erweitert. Diese Stile werden hier aber nicht beschrieben. In den Beispielen zum Buch finden Sie in der Datei FilterStile.htm eine Demo für die recht interessanten Filter-Stile des Internet Explorers.

5.2 Die drei Möglichkeiten, CSS einzusetzen

CSS-Stile können prinzipiell mit drei verschiedenen Techniken verwendet werden: Inline, eingebettet und eingebunden.

Dieser Abschnitt beschreibt zunächst die Techniken zur Verwendung von Stilen. Die einzelnen Stile werden hier nicht beschrieben, diese finden Sie ab Seite 112.

5.2.1 Inline-Stile

Sie können den Stil jedes einzelnen HTML-Element in seinem Start-Tag definieren. Die Syntax dazu ist:

`<`*Tagname* `[`*Attribute*`] style="`*Stilangabe*`">`

In der Stilangabe geben Sie die zu verwendenden Stile an. Dabei schreiben Sie zuerst den Namen des Stils, dann einen Doppelpunkt und schließlich den oder die Werte, die Sie diesem Stil zuweisen wollen. Wenn Sie einem Stil mehrere Werte zuweisen wollen, trennen Sie die einzelnen Werte durch ein Leerzeichen. Mehrere Stile trennen Sie durch Semikola. Das folgende Beispiel weist einem Header-Tag einen dünnen, durchgezogenen Rahmen und eine graue Hintergrundfarbe zu:

```
<h1 style="border: thin solid;
 background-color:silver">
 Das ist eine Überschrift mit Stil
</h1>
```

5.2.2 Eingebettete Stile

Wenn Sie den Standard-Stil von HTML-Tags grundsätzlich ändern wollen oder Gruppen von Elementen mit gleichen Stilen versehen wollen, können (bzw. sollten) Sie die Stile im Kopfbereich des Dokuments in einem `style`-Tag definieren. Die Syntax zur Anpassung der Stile von Tag ist:

```
<style type="text/css">
 Tagname {Stilangabe}
 Tagname {Stilangabe}
 ...
</style>
```

Tagnamen geben Sie ohne die spitzen Klammern an. Die Stilangaben werden genauso angegeben wie bei Inline-Stilen. Das folgende Beispiel definiert die Stile der Tags `<h1>`, `<h2>`, `<h3>` und `<p>`:

```
<head>
 ...
<style type="text/css">
 <!--
 body {background-color:silver}
 h1 {font: bold 18pt Arial; color: black;
  background-color: white; border: thin solid red}
 h2 {font: bold 14pt Arial; color: black;
  background-color: white; border: thin solid blue}
 h3 {font: bold 12pt Arial; color: black;
  background-color: white; border: thin solid green}
 p {font: 10pt Arial; color: white}
 -->
</style>
...
</head>
```


Das Ergebnis dieses Quellcode sehen Sie in Abbildung 5.2 auf Seite 111.

Die HTML-Kommentare im `style`-Tag sorgen übrigens dafür, dass Browser, die CSS nicht kennen, nicht die enthaltenen Stilbeschreibungen im Dokument anzeigen.

Die Kommentare dürfen keinen weiteren Text beinhalten. Wenn Sie Text im Kommentar unterbringen, können Sie nicht davon ausgehen, dass die Stile in allen Browsern korrekt ausgewertet werden. Beim Netscape 4.7 und 6.1 funktionieren Stile z. B. nicht korrekt, wenn der Kommentar Text enthält. Daraus folgt auch, dass der `style`*-Tag keine HTML-Kommentare enthalten darf.*

Werden im HTML-Dokument nun die so umdefinierten Tags verwendet, zeigt der Browser diese HTML-Elemente mit den definierten Stilen an.

Stil-Klassen

Neben der Anpassung des Stils von Tags können Sie noch Klassen von Stilen definieren. Eine Stil-Klasse ist zunächst nicht mit einem Tag verbunden. Erst wenn Sie die Stil-Klasse im `class`-Attribut eines Tags angeben, weisen Sie dem entsprechenden HTML-Element den in dieser Klasse definierten Stil zu. Die Syntax zur Erzeugung von Stil-Klassen ist:

```
<style>
 <!--
 .Klassenname {Stilangabe}
 .Klassenname {Stilangabe}
 ...
 -->
</style>
```

Der folgende Quellcode erzeugt eine Stil-Klasse für Zitate und eine für Code:

```
<style type="text/css">
 <!--
 .cite {font: italic 12pt Times; color: yellow}
 .code {font: 10pt Courier; color: green}
 -->
</style>
```

HTML-Elemente, die dieser Klasse angehören, besitzen dann den in der Stil-Klasse definierten Stil:

```
<p class="cite">
 Das ist ein Absatz der Klasse 'cite'
</p>
<span class="cite">
 Das ist ein Span der Klasse 'cite'
</span>
<p class="code">
 Das ist ein Absatz der Klasse 'code'
</p>
```

Stil-Klassen können Sie auch mit Tags verbinden. Der Stil wird dann nur für HTML-Elemente mit demselben Tag angewandt, die der angegebenen Klasse angehören:

```
<style type="text/css">
 <!--
 h1 {font: bold 18pt Arial; color: black;
  background-color: white; border: thin solid red}
 h1.special {font: bold 24pt Arial; color: black;
  background-color: white; border: thin solid red}
 -->
</style>
...
<h1 class="special">
 Überschrift mit speziellem Stil der Klasse 'special'
</h1>
<h1>
 Normale Überschrift
</h1>
```


Stile für Elemente mit bestimmten IDs

Schließlich können Sie Stile noch so definieren, dass diese nur auf Elemente angewandt werden, die eine bestimmte ID besitzen:

```
<style type="text/css">
 <!--
 #myid {font: bold italic 12pt Times; color: blue}
 -->
</style>

<div id="myid">Das ist ein Div mit der ID 'myid'.</div>
<div id="myid">Das ist ein Div mit der ID 'myid'.</div>
<div id="otherid">Das ist ein Div mit der ID
 'otherid'.</div>
```

Das Ergebnis

Abbildung 5.2 zeigt das Ergebnis der vorhergehenden eingebetteten Stil-Beispiele.

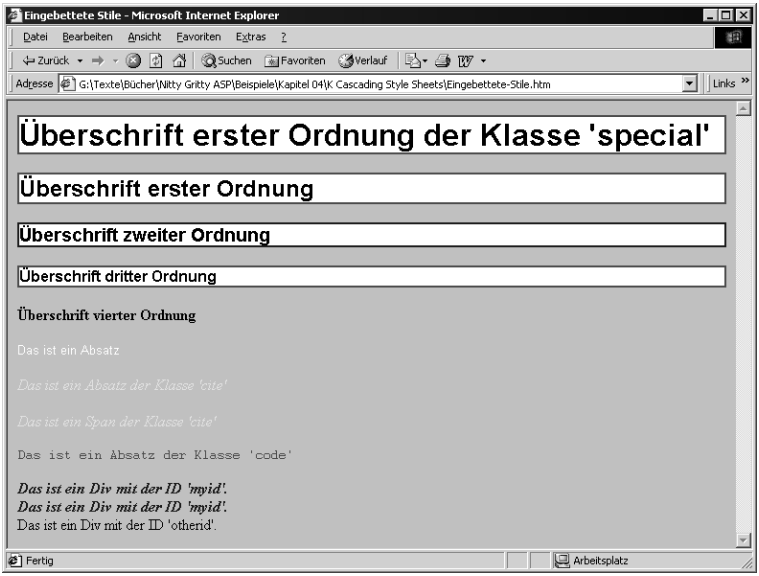


Bild 5.2: Eine HTML-Seite mit eingebetteten Stilen im Internet Explorer

5.2.3 Eingebundene Stile

Statt Stile in jedem HTML-Dokument separat zu definieren, können Sie diese auch in einer separaten Datei verwalten und in die einzelnen Dokumente einer Website einbinden. In der CSS-Datei werden die Stile wie bei eingebetteten Stilen definiert, nur dass der `style`-Tag wegfällt:

```
@charset "iso-8859-1";

body {background-color:silver}
h1 {font: bold 18pt Arial; color: black;
 background-color: white; border: thin solid red}
h2 {font: bold 14pt Arial; color: black;
 background-color: white; border: thin solid blue}
h3 {font: bold 12pt Arial; color: black;
 background-color: white; border: thin solid green}
p {font: 10pt Arial; color: white}
```

Den zu verwendenden Zeichensatz geben Sie über die Zeile `@charset "iso-8859-1"` an. Die Bedeutung des korrekt angegebenen Zeichensatzes ist dieselbe wie bei HTML.

Im HTML-Dokument binden Sie die CSS-Datei nun über einen Link ein:

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

Der unübersehbare Vorteil solcher eingebundenen Stildateien ist, dass eine Änderung der Stile in dieser Datei sich sofort auf *alle* Dokumente auswirkt, die die Stildatei einbinden. Damit können Sie den Stil einer ganzen Website sehr flexibel und einfach verwalten (abgesehen von den recht komplexen Stilangaben).

5.3 Kaskadierende Stile

Das »Cascading« in den Cascading Style Sheets sagt aus, dass Stile auch kaskadierend aufeinander aufgebaut werden können. Wenn einem Tag z. B. in einer eingebundenen Style-Sheet-Datei die Schriftgröße 12 zugewiesen wurde, und diesem Tag wird innerhalb des Dokuments zusätzlich eine fette Schriftauszeichnung vergeben, besitzen alle Elemente dieses Tags innerhalb des Dokuments die Schriftgröße 12 *und* eine fette Auszeichnung. Stile können also aufeinander aufbauen. Falls in einer Stilbeschreibung ein Stil neu definiert wird, der bereits zuvor definiert wurde, gilt immer der neuere Stil.

5.4 Die wichtigsten CSS-Stile

Die Cascading Style Sheets definieren eine große Anzahl an Stilen. Viele davon können auf die meisten HTML-Tags angewendet werden, einige funktionieren (meistens naturgemäß) mit einigen Tags nicht. Die Beschreibung aller Stile, und deren Verwendbarkeit mit den verschiedenen Tags, würde den Rahmen dieses Buchs sprengen. Ich be-

schreibe deshalb hier nur die in meinen Augen wichtigsten Stile. Im Online-Artikel »Die Stile der Cascading Style Sheets« finden Sie die anderen wichtigen Stile beschrieben.

Eine kurze und gute Übersicht über die einzelnen Stile finden Sie auf der Seite http://www.webreview.com/style/css1/glossary.shtml.

5.4.1 Ein Tipp zuvor

Wenn Sie mit Stilen arbeiten, sollten Sie immer überprüfen, ob Ihre Dokumente auch in allen gängigen Browsern funktionieren. Dazu gehören natürlich der Internet Explorer 5.x und 6.x, der Netscape 4.x (der immer noch von vielen Benutzern eingesetzt wird, weil die neuen Version 6.x wenig Zuspruch findet) und die neue Netscape-Version 6.x. Problematisch ist leider, dass der IE nicht gleichzeitig in mehreren Versionen installiert werden kann (beim Netscape-Browser geht das schon). Testen Sie einmal die Beispieldateien dieses Kapitels in den verschiedenen Versionen der Browser, und Sie sehen, was ich meine. Besonders die Seite Positions- und Grössen-Stile.htm wird vom Netscape 4.7 sehr konfus dargestellt.

5.4.2 Maßeinheiten

Wenn Sie in einem Stil eine Breite, Höhe oder andere Größenangabe machen müssen, können Sie eine der in Tabelle 5.1 und Tabelle 5.2 beschriebenen Maßeinheiten verwenden.

Absolute Einheiten

Absolute Einheiten geben die Größe absolut an.

Einheit	Beschreibung
pt	Punkt. Ein Punkt entspricht 1/72 Inch.
pc	Pica. Ein Pica entspricht 12 Punkten.
in	Inch (oder auch Zoll). Ein Inch entspricht 2,54 cm.
mm	Millimeter
cm	Zentimeter

Tabelle 5.1: Die absoluten Maßeinheiten der Cascading Style Sheets

Relative Einheiten

Relative Einheiten geben die Größe relativ zu anderen Größen an.

Einheit	Beschreibung
`em`	`em` bezieht sich auf die Höhe der Schriftart, die dem Element zugewiesen ist. `style="width:7em"` bedeutet z. B.: 7-mal so breit wie ein Zeichen hoch ist. Ändert sich bei diesem Beispiel die Schriftgröße, ändert sich auch die Breite des Elements.
`ex`	`ex` verhält sich wie `em`, nur dass hier die Höhe des kleinen x als Basis verwendet wird.
`px`	Pixel (die Größe eines Pixel ist relativ zur verwendeten Auflösung und zum Ausgabegerät).
`%`	Prozent. Eine Prozentangabe verhält sich relativ zur Größe des übergeordneten Elements, in dem das Element eingebettet ist.

Tabelle 5.2: Die relativen Maßeinheiten der Cascading Style Sheets

5.4.3 Wichtige Stile

Die folgende Auflistung beschreibt einen kleinen, aber wichtigen Ausschnitt der CSS-Stile:

- `font-family:`*family*: definiert die Schriftart. Sie können mehrere Schriftarten angeben, die Sie mit Kommata trennen.
- `font-size:`*size*: Dieser Stil legt die Schriftgröße in den üblichen Einheiten (pt, % etc.) oder in einer speziellen »ungenauen« Einheit (xx-small, x-small, small, medium, large, x-large, xx-large, smaller) fest. Eine prozentuale Angabe bezieht sich auf die normale Größe der Schrift.
- `font-weight:`*weight*: Mit diesem Stil definieren Sie, ob und wie fett die Schrift ausgegeben wird. Sie können die folgenden Werte verwenden: `bold` (fett), `bolder` (extra fett), `lighter` (dünner). Der Internet Explorer und der Netscape unterscheiden `bold` und `bolder` nicht.
- `font-style:`*style*: Definiert, ob die Schrift kursiv ausgegeben wird. Sie können die folgenden Werte verwenden: `italic`, `normal`.

- `font:[style] [variant] [weight] [size] [height] [family]`: Mit diesem Stil können Sie die einzelnen Schriftwerte zusammenfassen.
- `text-decoration:decoration`: legt die Dekoration eines Textes mit den folgenden Werten fest: `underline` (unterstrichen), `overline` (überstrichen), `line-through` (durchgestrichen). Der Wert `blink` ist im Standard vorgesehen, wird aber weder vom Internet Explorer noch vom Netscape unterstützt. Sie können mehrere Angaben kombinieren.
- `text-align:align`: bestimmt die horizontale Ausrichtung von Text mit den Werten `left`, `right`, `center` oder `justify` (Blocksatz).
- `color:Farbe`: `color` bestimmt die Text- und die Rahmenfarbe als RGB-Wert oder über eine der vordefinierten Farbangaben ("red", "green" etc.).
- `background-color:color`: Hintergrundfarbe als RGB-Wert oder eine der vordefinierten Farbangaben ("red", "green" etc.). Beispiel: `background-color:#003366`
- `background-image:url([Dateiname])`: Angabe eines Hintergrundbilds. Das Bild wird normalerweise so oft wiederholt, bis der Inhalt des HTML-Elements ausgefüllt ist. Mit dem Stil `background-repeat` können Sie dieses Verhalten jedoch auch beeinflussen.
- `background-repeat:repeat`: Mit dieser Angabe können Sie festlegen, ob und wie ein Hintergrundbild wiederholt ausgegeben wird. Verwenden Sie dazu die Werte `repeat` (horizontal und vertikal wiederholen), `repeat-x` (horizontal wiederholen), `repeat-y` (vertikal wiederholen) oder `no-repeat` (nicht wiederholen),
- `background-attachment:scroll | fixed`: Ein Hintergrundbild wird normalerweise mitgescrollt, wenn der Inhalt des HTML-Elements gescrollt wird. Mit `background-attachment:fixed` können Sie erreichen, dass das Hintergrundbild stehen bleibt.
- `background:[color] [image] [repeat] [attachment] [position]`: Mit diesem Stil können Sie die Hintergrund-Stile zusammenfassen. Die Reihenfolge der Angaben muss eingehalten werden. Einzelne Angaben können Sie allerdings auch einfach weglassen.
- `border-width:width`: Die Breite des Rahmens um ein HTML-Element mit den Werten `thin` (dünn), `medium` (mittel) und `thick` (dick).

- `border-style:`*style*: Dieser Stil steuert die Art des Rahmens um ein HTML-Element. Die Einstellungen sind `dotted` (gepunktet), `dashed` (gestrichelt), `solid` (durchgezogen), `double` (doppelt gerahmt), `groove` (vertiefter 3D-Rahmen), `ridge` (erhöhter 3D-Rahmen), `inset` (vertieft) und `outset` (erhöht).
- `border-color:`*color*: definiert die Farbe des Rahmens.
- `border: [`*width*`] [`*style*`] [`*color*`]`: Zusammengesetzte Angabe der Rahmeneigenschaften. Beispiel: `border: thin solid black`.
- `margin:` *margins*: Zusammengesetzte Angabe der Ränder. Wenn Sie nur einen Wert angeben, gilt dieser für aller Ränder. Geben Sie zwei Werte ein, gilt der erste Wert für den oberen und unteren Rand und der zweite für den linken und rechten. Vier angegebene Werte gelten in der folgenden Reihenfolge: oben, rechts, unten und links.
- `padding:` *paddings*: Zusammengesetzte Angabe der inneren Ränder. Die Werte werden wie bei `margin` ausgewertet.
- `position:` *position*: Dieser CSS2-Stil bestimmt die Art der Positionierung eines Elements mit den folgenden Werten: `static` (die Positionierung wird vom Browser bestimmt), `relative` (die Position ergibt sich aus der Standardposition addiert mit den in `left` und `top` angegebenen Werten); `absolute` (die Position wird durch die in `left` und `top` angegebenen Werte bestimmt), `fixed` (wie absolute, nur dass das Element fixiert ist, also nicht gescrollt wird, falls Scrollen möglich ist) und `inherit` (die Art der Positionierung wird vom übergeordneten Element geerbt).
- `left:` *left*; `top:` *top*; `right:` *right*; `bottom:` *bottom*: Mit diesen CSS2-Stilen können Sie die Position bestimmen, wenn der `position`-Stil eine Positionierung erlaubt.
- `width:` *width*; `height:` *heigth*: Mit diesem Stilen können Sie die Breite und Höhe eines Elements bestimmen.

6 DHTML

Auf der Clientseite (im Browser) können Sie mit JavaScript oder VB-Script Programme schreiben, die über das Objektmodell des Browsers auf die HTML-Elemente des Dokuments zugreifen und diese dynamisch verändern können. Das ist DHTML. Menüs, die erst dann aufklappen, wenn der Anwender mit der Maus darüber fährt, oder Links, die ihre Hintergrundfarbe bei Mausberührung wechseln, sind meist über DHTML programmiert. Dieses Kapitel erläutert, was DHTML ist und wie Sie mit JavaScript auf die Elemente einer Website zugreifen und auf Ereignisse wie Mausbewegungen reagieren können.

*DHTML wird in diesem Kapitel nur grundlegend erläutert. Die Arbeit mit den verschiedenen Browsern kann sehr umfangreich und kompliziert werden. Wenn Sie die Möglichkeiten von DHTML erforschen wollen, schauen Sie einmal bei Dynamic Drive vorbei (*www.dynamicdrive.com*). Dort finden Sie sehr viele, teilweise ziemlich coole DHTML-Beispiele, die meist in allen wichtigen Browsern laufen. Da Sie die Quellcodes oft zur freien Verfügung downloaden können, können Sie Ihre Dokumente sehr einfach mit DHTML-Features ausrüsten. Ein komplexes Menü würde ich z. B. nicht selbst entwickeln, sondern einfach eines von Dynamic Drive downloaden …*

JavaScript behandelt dieses Kapitel nicht, weil dies den Rahmen des Buchs sprengen würde. Diese Sprache wird ja auch im JavaScript-Buch der Nitty-Gritty-Reihe ausführlich behandelt. Ich gehe in den Beispielen also davon aus, dass Sie die Grundlagen von JavaScript kennen. Alternativ finden Sie aber auch eine kurze JavaScript-Beschreibung im Online-Artikel »JavaScript-Grundlagen« (oder auf einer der vielen JavaScript-Websites, Links dahin finden Sie auf der Website zum Buch).

6.1 Was ist DHTML?

DHTML (Dynamic HTML) ist keine eigene Sprache oder Technik. DHTML setzt sich aus HTML, JavaScript, CSS und dem Document Object Model (DOM) des Browsers zusammen. HTML und CSS kennen Sie bereits, JavaScript ist »nur« eine weitere Programmiersprache. Neu und umfangreich ist das Document Object Model. Mit JavaScript (im IE auch mit VBScript) greifen Sie über das DOM auf die Elemente einer HTML-Seite zu. Dazu reagieren Sie normalerweise auf Ereignisse, die der Browser für die einzelnen Tags zur Verfügung stellt. Der folgende (zunächst nur im Internet-Explorer funktionierende) Quelltext ändert z. B. die Hintergrundfarbe eines Textes, wenn die Maus diesen Text berührt:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
<title></title>

<script language="JavaScript">
 function changeColor(color)
 {
    document.all.hello.style.backgroundColor = color;
 }
</script>

</head>
<body>

<h1>DHTML-Demo</h1>

<span id="hello" name="hello"
 style="color:white; background-color:green;
  border:thin solid black"
  onmouseover="changeColor('red');"
  onmouseout="changeColor('green');">
Hello DHTML
</span>

</body>
</html>
```

Im Prinzip ist DHTML (wenn Sie JavaScript kennen) einfach. Viel Wissen erfordert aber das Objektmodell des Browsers. Hinzu kommt, dass die Objektmodelle der verschiedenen Browser sich teilweise erheblich unterscheiden.

Das Document Object Model (DOM)

Jeder DHTML-fähige Browser stellt zum Zugriff auf die Objekte einer HTML-Seite ein DOM zur Verfügung. Das DOM ist eine Objekthierarchie, die mit dem `window`-Objekt beginnt, über das Sie Zugriff auf das Browserfenster erhalten. Das untergeordnete `document`-Objekt repräsentiert das geladene Dokument. Über dieses Objekt können Sie alle HTML-Elemente bearbeiten und sogar neue HTML-Elemente erzeugen.

Das DOM ist mittlerweile vom W3C standardisiert (*www.w3.org/DOM/*). Leider wird es zurzeit nur von wenigen Browsern überhaupt implementiert (so weit mir bekannt ist nur vom Netscape und vom IE). Erschwerend kommt hinzu, dass Netscape und Microsoft ihr DOM (natürlich) bis zur Version 6.x nicht kompatibel implementiert haben und bei älteren Browserversionen in Teilen vom W3C-Standard abweichen. Die Versionen 6.x von Netscape und Microsoft unterstützen den W3C-Standard aber (angeblich) vollständig. In Ihren DHTML-Programmen müssen Sie also immer wieder überprüfen, welcher Browser benutzt wird, und je nach Browser spezifischen Programmcode erzeugen. Das macht die DHTML-Programmierung nicht gerade einfach.

Die Objekte des Netscape Browsers werden in der JavaScript-Referenz von Netscape auf der Seite developer.netscape.com/docs/manuals/js/client/jsref/Index.htm *beschrieben. Klicken Sie auf den Link* Table of Contents, *um zum Inhaltsverzeichnis mit der Auflistung der Objekte zu gelangen. Microsoft beschreibt die Objekte des Internet Explorers auf der Seite* msdn.microsoft.com/workshop/author/dhtml/reference/dhtmlrefs.asp.

6.2 Arbeiten mit dem DOM

Das Document Object Model (DOM) des Browers ist für die Arbeit mit DHTML so ziemlich das wichtigste Werkzeug. Leider unterscheiden sich die DOMs der verschiedenen Browser, wie ich dies ja bereits erläutert habe. Ich beschreibe deshalb hier, wie Sie prinzipiell mit dem DOM arbeiten, und erläutere die wichtigsten Objekte und deren wichtigste Eigenschaften und Methoden.

6.2.1 Grundlegendes

Erst der Internet Explorer und der Netscape in der Version 6.x unterstützen das vom W3C standardisierte DOM komplett. Der Netscape 4.x bietet nur ein erheblich eingeschränktes Objektmodell, über das Sie nur einen Bruchteil der HTML-Elemente ansprechen können. DHTML wird im Netscape 4.x auch recht kompliziert über Netscape-eigene `layer`-Tags realisiert, die aber nur wenig Features (wie dynamische Positionierung) ermöglichen. Der Internet Explorer ab Version 4.x ermöglicht bereits den Zugriff auf alle HTML-Elemente und deren Eigenschaften, verwendet dafür aber eine vom Netscape-Browser etwas abweichende Syntax.

In allen DOMs beginnt das Objektmodell mit dem `window`-Objekt. Dieses Objekt repräsentiert das Browser-Fenster. Das `document`-Objekt, welches das geladene HTML-Dokument darstellt, ist dem `window`-Objekt direkt untergeordnet. Dieses Objekt enthält einige Auflistungen, über die Sie die Elemente des HTML-Dokuments erreichen. Über die `all`-Auflistung des Interner Explorers erhalten Sie z. B. Zugriff auf die HTML-Element der Seite. Den Text eines Elements mit dem Namen »hello« ändern Sie im IE so:

```
window.document.all.hello.innerText = "Hallo Benutzer";
```


Um auf das `document`*-Objekt zuzugreifen, müssten Sie normalerweise immer* `window.document` *schreiben. Das* `window`*-Objekt wird jedoch implizit vor eine Objektangabe gesetzt, wenn es nicht angegeben ist. Sie können also auch direkt mit* `document` *auf das* `document`*-Objekt zugreifen.*

```
document.all.hello.innerText = "Hallo Benutzer";
```

Tabelle 61. und6.2 zeigen die wichtigsten Eigenschaften und Methoden des `window`-Objekts. Browserspezifische Elemente sind mit »NS« (Netscape) bzw. »IE« (Internet Explorer) gekennzeichnet.

Eigenschaft	Beschreibung
`document`	verweist auf das `document`-Objekt
`history`	verweist auf ein `history`-Objekt, über das Sie zu einer bestimmten URL springen können. Die `back`-Methode geht zur vorherigen URL, die `forward`-Methode zur nächsten (falls möglich)

Tabelle 6.1: Die wichtigsten Eigenschaften des window-Objekts

Methode	Beschreibung
`alert`([*Message*])	zeigt eine Warnmeldung an
`close`()	schließt das Fenster
`open`([*URL*][, *Name*] [, *Features*][, *Replace*])	öffnet eine URL im Browserfenster. Wenn *Name* nicht angegeben ist, wird ein neues Fenster geöffnet. Im *Name*-Argument geben Sie einen Wert für das `target`-Attribut eines Links an. "_self" sorgt z. B. dafür, dass die URL in dem Fenster geöffnet wird, aus dem heraus der Aufruf der Methode erfolgt. In *Features* können Sie einige Features des Fensters angeben. "height=500" definiert die Höhe, "width=700" die Breite, "titlebar=no" definiert, dass das Fenster keine Titelleiste besitzt.
`print`()	druckt das aktuelle Dokument aus
`prompt`([*Message*] [, *DefaultValue*])	zeigt einen einfachen Eingabedialog an und gibt den eingegebenen Wert zurück

Tabelle 6.2: Die wichtigsten Methoden des window-Objekts

Tabelle 6.3 beschreibt die wichtigsten Eigenschaften des `document`-Objekts, listet die wichtigsten Methoden auf.

Eigenschaft	Beschreibung
`all` (IE)	eine Auflistung der HTML-Elemente der Seite, die einen Namen besitzen
`anchors`	eine Auflistung aller a-Tags, die einen Namen besitzen
`bgColor`	die Hintergrundfarbe
`fgColor`	die Vordergrundfarbe
`forms`	eine Auflistung aller Formulare der Seite
`frames` (IE)	eine Auflistung der Frames, die im aktuellen Dokument enthalten sind
`images`	eine Auflistung der `img`-Elemente der Seite
`links`	eine Auflistung aller a-Tags der Seite, die ein `href`-Attribut besitzen
`layers` (NS)	eine Auflistung der Netscape-eigenen Layer-Elemente (siehe Seite 127)

Tabelle 6.3: Die wichtigsten Eigenschaften des document-Objekts

Methode	Beschreibung
`write`*(Text)*	schreibt einen Text in das Dokument. Beachten Sie, dass `write` (wie `writeln`) das Dokument löscht, wenn Sie diese Methode nicht *während*, sondern *nach* der Erzeugung eines Dokuments anwenden.
`writeln`*(Text)*	schreibt einen Text in das Dokument, gefolgt von einem Carriage Return (allerdings nur im Quelltext, `writeln` erzeugt kein ` `).

Tabelle 6.4: Die wichtigsten Methoden des document-Objekts

Die Eigenschaften der einzelnen HTML-Elemente sind oft identisch. Tabelle 6.5 zeigt deshalb die wichtigsten. Eine sehr wichtige Eigenschaft ist `style`. `style` verweist auf ein Objekt, über das Sie den Stil des Elements abfragen und verändern können. Die Eigenschaften dieses Objekts sind prinzipiell identisch mit den Stilnamen. Stile, deren Name einen Bindestrich enthält, werden allerdings ohne Bindestrich benannt. Die einzelnen Worte werden dann mit einem Groß-

buchstaben begonnen. Der Stil `background-color` wird z. B. über die Eigenschaft `backgroundColor` repräsentiert.

Eigenschaft	Beschreibung
`innerHTML` (IE)	der HTML-Text des Elements
`innerText` (IE)	der reine Text des Elements
`style.backgroundColor`	Hintergrundfarbe
`style.backgroundImage`	URL des Hintergrundbilds
`style.borderColor`	Farbe des Rahmens
`style.borderStyle`	Stil des Rahmens ("dotted", "dashed", "solid" etc.)
`style.borderWidth`	Breite des Rahmens ("medium", "thin", "thick", genaue Breitenangabe)
`style.color`	Vordergrundfarbe
`style.font`	Schriftart
`style.left`	linke Position des Elements
`style.pixelHeight`	Höhe des Elements
`style.pixelWidth`	Breite des Elements
`style.top`	obere Position des Elements
`style.zIndex`	der Index des Elements innerhalb der Ebenen des Dokuments

Tabelle 6.5: Die wichtigsten Eigenschaften der HTML-Elemente

6.2.2 DHTML im Internet Explorer 4 und 5

Im Internet Explorer können Sie ab der Version 4 auf alle Elemente einer HTML-Seite zugreifen (was beim Netscape 4.x nicht der Fall ist). Neben den Auflistungen `forms`, `images` und `links`, die alle Browser kennen, ist beim IE die `all`-Auflistung wichtig. Über diese Auflistung greifen Sie auf beliebige HTML-Elemente zu. Sie können den Namen des Elements über die allgemeine Objektsyntax direkt anhängen:

`document.all.`*`Elementname`*

Alternativ können Sie die Auflistung auch als solche verwenden, indem Sie den Namen als String-Index angeben:

```
document.all["Elementname"]
```

Über diese Syntax können Sie auf die Eigenschaften der Elemente zugreifen oder deren Methoden aufrufen. Der folgende Quelltext ändert z. B. den Text eines `span`-Elements mit dem Namen *hello*:

```
document.all.hello.innerText = "This is DHTML";
```

Wenn das Element auf einem Formular angelegt ist, müssen Sie den Formularnamen mit angeben. Die folgende Zeile ermittelt den Wert einer Textbox auf einem Formular:

```
alert(document.all.frmInput.txtLogin.value);
```

Ein wichtiger Trick ist, dass Sie `all` dann auch weglassen können:

```
alert(document.frmInput.txtLogin.value);
```


Der Trick ist deshalb wichtig, weil auch der Netscape diese Syntax unterstützt. Das Cross-Browser-Coding wird damit wenigstens für Formularauswertungen wesentlich einfacher.

Die Arbeit mit den anderen Auflistungen des IE ist identisch. Interessant sind diese Auflistungen, wenn Sie in einer Schleife alle Elemente eines bestimmten Typs durchgehen wollen. Das folgende Beispiel geht alle `img`-Elemente durch und definiert für diese Elemente einen Rahmen und eine Hintergrundfarbe:

```
var i;
for (i=0; i<document.images.length; i++)
{
   document.images[i].border = "1";
   document.images[i].style.backgroundColor = "black";
}
```

6.2.3 Netscape 4

Der Netscape 4 repräsentiert nur sehr wenige HTML-Elemente in seinem Objektmodell. DHTML wird hier eigentlich ausschließlich über eigene `layer`-Elemente programmiert. Der Netscape 4 lässt allerdings auch einen Zugriff auf Steuerelemente zu, die auf einem Formular angelegt wurden.

Layer zur dynamischen Positionierung

Innerhalb eines Layer platzieren Sie andere HTML-Elemente oder Text. Bei diesen Elementen können Sie prinzipiell nur die Sichtbarkeit und Position und einige wenige Attribute des Layer (wie z. B. die Hintergrundfarbe) einstellen. Layer, die reinen Text enthalten, können Sie aber auch mit neuem Text beschreiben. Leider unterstützt der Netscape 4.x die wichtigen Ereignisse `onMouseOver`, `onMouseOut` und `onMouseMove` nur für Layer-Elemente, nicht für Spans und Divs.

Layer deklarieren Sie mit Hilfe des `layer`-Tag. Netscape arbeitet nicht mit dem `id`-Attribut zur Benennung des Layers, sondern mit `name`:

```
<layer name="layer1">Layer 1</layer><br>
```

Der Text eines Layers kann nicht wie beim Internet Explorer über die Eigenschaften `innerText` und `innerHTML` verändert werden, weil es diese Eigenschaften nicht gibt. Um Text in einem Layer auszugeben, müssen Sie die `write`-Methode des `document`-Objekts des Layers verwenden (jeder Layer besitzt ein eigenes `document`-Objekt):

```
document.layers.layer1.document.write("Neuer Text");
document.layers.layer1.document.close();
```

Das »Schließen« des Dokuments ist notwendig, weil der Browser den Text erst dann schreibt.

Die Sichtbarkeit eines Layers steuern Sie über dessen `visibility`- Eigenschaft:

```
<layer name="layer2" visibility="hide">Layer 2</layer>
```

In JavaScript setzen Sie diese Eigenschaft dann auf den Wert "show", wenn Sie den Layer anzeigen wollen:

```
document.layers.layer2.visibility = "show";
```

Die Position eines Layers wird über die Eigenschaften `left` und `top` definiert:

```
<layer name="layer3" id="Layer3" top="400" left="10">
 Layer 3</layer>
```

Über diese Eigenschaften oder über die `moveTo`-Methode können Sie die Position dann verändern:

```
document.layers.layer3.left = 100;
document.layers.layer3.left.moveTo(100, 550);
```

Ein häufig benutzter Trick ist, dass der Netscape 4.x einen `div`- oder `span`-Tag mit dem Stil "position:absolute" fast wie einen Layer behandelt. Damit werden DHTML-Seiten einfacher, die in mehreren Browser-Varianten arbeiten:

```
<div name="div1" id="div1" style="position:absolute">
 Div 1</div><br>
<span name="span1" id="span1" style="position:absolute">
 Span 1</span>
```

Zugriff auf Steuerelemente

Der Netscape 4 erlaubt wie der Internet Explorer den Zugriff auf Steuerelemente, die auf einem Formular angelegt sind. Verwenden Sie dazu einfach die folgende Syntax:

```
document.Formulaname.Steuerelementname
```

Die HTML-Atribute der Steuerelemente werden in Eigenschaften repräsentiert. Den Wert einer Textbox »txtLogin«, die auf einem Formular »frmInput« angelegt ist, fragen Sie so ab:

```
alert(document.frmInput.txtLogin.value);
```

Tipps

JavaScript-Fehler »xyz has no properties«

Dieser Fehler wird oft erzeugt, wenn Sie einen Bezeichner als Objekt verwenden, der gar kein Objekt bezeichnet. Beim Umstellen von Internet-Explorer-Skripten habe ich öfter `document.all` verwendet. `all` existiert aber nicht im Netscape. Da JavaScript nicht existierende Variablen einfach implizit deklariert, wird `all` in diesem Fall einfach als Variable erkannt, die aber eben kein Objekt ist und deshalb auch keine Eigenschaften besitzt. Derselbe Fehler wird auch erzeugt, wenn Sie korrekterweise `document.layers` verwenden, als Element dann aber ein HTML-Element angeben, das gar kein Layer ist (z. B. einen Span ohne den Stil »position:absolute«).

6.2.4 Internet Explorer ab Version 5.5 und Netscape 6.x

Ab Netscape 6.0 und Internet Explorer 5.5 greifen Sie über die im W3C-DOM standardisierte `document`-Methode `getElementById` auf alle benannten Tags der HTML-Seite zu. Beide Browser unterstützen, wie auch der Internet Explorer 5.0, die wichtige `style`-Eigenschaft für HTML-Elemente. Das folgende Beispiel macht einen Span sichtbar und definiert danach dessen Position neu:

```
with (document.getElementById("span1"))
{
   style.visibility = "visible";
   style.left = "50px";
   style.top = "450px";
}
```

Die Eigenschaften der HTML-Elemente und des `style`-Objekts sind wie beim Internet Explorer 5.0 benannt: der Name der Eigenschaft entspricht jeweils dem korrespondierenden HTML-Attribut bzw. CSS-Stils und wird (mit Ausnahmen bei zusammengesetzten Stil-Namen) grundsätzlich kleingeschrieben. Das HTML-Attribut `value` wird z. B. über die Eigenschaft `value` repräsentiert. Stile, deren Name einen Bindestrich enthält, werden ohne Bindestrich benannt. Die einzelnen Worte beginnen dann mit einem Großbuchstaben. Der Stil `background-color` wird über die Eigenschaft backgroundColor repräsentiert.

Die Eigenschaften `innerText` und `innerHTML` zur Bearbeitung des Inhalts eines HTML-Elements gehören nicht zum W3C-Standard und werden deshalb nur vom Internet Explorer unterstützt. Sie können diese Eigenschaft aber auch verwenden, wenn die Seite im Netscape angezeigt werden soll. Da JavaScript Variablen (und Eigenschaften) einfach neu deklariert, wenn diese nicht deklariert sind, führt die Verwendung dieser Eigenschaften nicht zu einem Fehler (aber leider auch nicht zu einem Ergebnis).

6.2.5 Cross-Browser-Coding

Wenn Sie Webseiten entwickeln, die in mehreren Browsern funktionieren, müssen Sie auf die Eigenheiten dieser Browser eingehen. Wenn Sie nur den Internet Explorer ab Version 5.5 und den Netscape

ab Version 6.0 berücksichtigen wollen, ist die Programmierung sehr einfach, weil diese beiden Browser den W3C-Standard unterstützen. Einfach ist auch, wenn Sie nur den Internet Explorer ab Version 4 unterstützen, weil die neuen Versionen kompatibel sind (`document.all` funktioniert auch in den neuen Versionen). Wollen Sie den Internet Explorer 4 bis 6 und den Netscape 6 berücksichtigen, müssen Sie eigentlich nur darauf achten, dass der IE 4-5.0 `document.all` verwendet (und dass der Netscape `innerText` und `innerHTML` nicht unterstützt). Mit dem Netscape 4 wird es aber schwierig, weil dieser Browser ja leider nur Layer mit ganz eigenen Attributen in DHTML kennt. Da der Netscape 4.x leider für Divs und Spans nicht die gängigen Ereignisse, wie z. B. `onMouseOver` und `onMouseOut`, unterstützt, ist DHTML in diesem Browser schon arg eingeschränkt. Leider verwenden viele Benutzer lieber die Version 4.5 oder 4.7 als die Version 6.0/6.1.

Wenn Sie im Scriptcode lediglich die Eingaben von Steuerelementen auswerten wollen, die auf einem Formular angelegt sind, müssen Sie nichts weiter machen. Verwenden Sie dazu einfach die folgende Syntax:

`document.`*`Formularname`*`.`*`Steuerelementname`*`.`*`Eigenschaft`*

Diese Syntax wird vom Netscape und vom Internet Explorer (so weit ich weiß ab Version 4) unterstützt. Die clientseitige Validierung von Benutzereingaben behandelt übrigens der Online-Artikel »ASP-Tipps und Tricks«.

Wenn Sie jedoch auf beliebige HTML-Elemente zugreifen wollen, müssen Sie normalerweise zunächst im Code herausfinden, welcher Browser verwendet wird. Sie können dazu die umständliche Methode wählen, indem Sie den Browsernamen und die Version über das `navigator`-Objekt abfragen:

```
browser = navigator.appName;
version = navigator.appVersion
```

Leider ist die Versionsangabe oft etwas kompliziert. Der Internet Explorer 6 gibt den (je nach Umfeld unterschiedlichen) String »4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR 1.03215)« zurück, der Netscape 6 den String »5.0 (Windows; de-DE)« (5.0 ist hier kein Schreibfehler). Die tatsächliche Version herauszufinden, ist also nicht allzu einfach.

Deshalb verwenden Internetprogrammierer lieber eine etwas andere Variante:

```
/* Browsertyp ermitteln */
var ns4 = false, ns6 = false, ie4 = false, ie6 = false;
if (document.all && document.getElementById)
   ie6 = true;
else if (document.getElementById)
   ns6 = true;
else if(document.all)
   ie4 = true;
else if (document.layers)
   ns4 = true;
```

Diese Variante überprüft einfach, ob die spezifischen `document`-Eigenschaften existieren.

Beachten Sie, dass diese vereinfachte Variante andere Browser wie z. B. Opera nicht berücksichtigt.

Das folgende einfache Beispiel setzt die Farbe eine HTML-Elements um, wenn der Anwender die Maus über das Element bewegt. Um das Netscape 4-Layer-Problem zu umschiffen, wende ich einen Trick an: Ich erzeuge einfach einen Layer, der mit dem `name`-Attribut bezeichnet wird, und einen `span`, der mit dem `id`-Attribut bezeichnet wird. Der Netscape 4.x verwendet den Layer und das `name`-Attribut, der Internet Explorer verwendet den `span` und das `id`-Attribut. Was der Netscape 6.x verwendet, kann ich nicht sagen. Es funktioniert aber ☺.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title></title>

<script language="JavaScript">
 /* Browsertyp ermitteln */
 var ns4 = false, ns6 = false, ie4 = false, ie6 = false;
 if (document.all && document.getElementById) ie6 = true;
 else if (document.getElementById) ns6 = true;
```

```
 else if(document.all)ie4 = true;
 else if (document.layers) ns4 = true;

 function changeColor(color)
 {
    /* Hintergrundfarbe des Span verändern */
    if (ie4)
       document.all.hello.style.backgroundColor = color;
    else if (ns4)
       document.layers.hello.bgColor = color;
    else if (ie6 || ns6)
       document.getElementById(
          "hello").style.backgroundColor = color;
}
</script>

</head>

<body>

<h1>DHTML-Cross-Browser-Demo</h1>

<!-- Ein Layer für den Netscape 4.x -->
<layer name="hello" left="10px"
 top="70px" bgColor="green"
 onMouseOver="changeColor('red');"
 onMouseOut="changeColor('green');">
<!-- Ein Span für den IE  -->
<div id="hello" style="position:absolute;
 left=10px;top=70px;background-color:green"
 onMouseOver="changeColor('red');"
 onMouseOut="changeColor('green');">
Hello Cross-Browser-DHTML
</div>
</layer>

</body>
</html>
```

Bei Ihren DHTML-Anwendungen sollten Sie beachten, dass Sie in ASP und ASP.NET den Browsertyp und die Version ermitteln können. Damit haben Sie die Möglichkeit, spezifischen DHTML-Code dynamisch zu erzeugen. Für den Netscape 4.x können Sie dann z. B. Layer-Elemente, für die anderen Browser Divs oder Spans erzeugen.

In den Buchbeispielen finden Sie ein HTML-Dokument mit einem kleinen Menü, das unter dem Netscape und dem Internet Explorer ab Version 4 funktioniert.

Teil II

Take that!

7 ASP-Grundlagen

Was ASP ist, wissen Sie ja bereits (wenn nicht, lesen Sie in Kapitel 1 noch einmal nach). Dieses Kapitel behandelt den grundlegenden Aufbau von ASP-Dokumenten, zeigt die Abarbeitung dieser Dokumente im IIS und führt Sie schließlich noch in die Arbeit mit VBScript, der bevorzugten ASP-Programmiersprache, ein.

Die Beispiele dieses Kapitels sind vorwiegend für ASP in VBScript entwickelt. Ich gehe bei Unterschieden jedoch auch auf ASP.NET ein. ASP.NET wird eigentlich erst ab Kapitel 11 behandelt. Dort finden Sie auch die Grundlagen zum .NET-Framework. Ich wollte aber bereits in diesem Kapitel auf Unterschiede zwischen ASP und ASP.NET hinweisen. In den Buchbeispielen finden Sie alle Quellcodes einmal für ASP und einmal für ASP.NET.

Ein echter Testserver für ASP

Wenn Sie Ihre ASP- und ASP.NET-Programme einmal in der realen Welt ausprobieren wollen, empfehle ich den Web-Hoster BRINKSTER (*www.brinkster.com*). Dort können Sie mit der »General Membership« 30 MB Platz auf dem Brinkster-Server nutzen. Und das für genau 0 Euro (bzw. 0 $). Brinkster unterstützt neben ASP 3 auch ASP.NET. Um ASP-Seiten auf den Server zu übertragen, nutzen Sie eine einfach zu bedienende Web-Oberfläche. Lesen Sie aber vorher die »ASP Rules«, die Sie auf der »File Manager«-Seite finden (die automatisch geöffnet wird, nachdem Sie sich eingeloggt haben).

7.1 Eine Anmerkung zuvor: Der IE-Cache

Wenn Sie mit ASP arbeiten, werden Sie Ihre Programme sicher auch testen wollen. Wenn Sie dazu den Internet Explorer verwenden, können Sie dabei in die Cache-Falle laufen. Damit Sie beim Testen keine Probleme haben, beschreibe ich hier zunächst, wie Sie diese Falle vermeiden.

Beim Internet Explorer müssen Sie beachten, dass dieser einmal geladene Dokumente in seinem Cache zwischenspeichert. Wenn Sie diese Dokumente erneut anfordern, liest der IE die Daten normalerweise nicht vom Webserver, sondern aus seinem Cache, sofern die Seite dort noch zu finden ist. Das führt häufig dazu, dass Sie ein ASP-Dokument ändern, danach erneut im IE anfordern, aber dieser zeigt die Änderungen nicht an. Probieren Sie das einmal aus: Erzeugen Sie ein Test-ASP-Dokument, das die aktuelle Zeit ausgibt (`Response.WriteTime`), öffnen Sie dieses im IE, fordern Sie im IE ein beliebiges anderes Dokument an und fordern das Testdokument dann im IE erneut an (über den Zurück-Schalter oder erneute Eingabe der URL): Das ASP-Dokument zeigt nicht die aktuelle Zeit, sondern die Zeit des ersten Abrufs an.

Dieses Problem existiert für dynamische ASP-Seiten auch für den späteren Benutzer. Fordert dieser dynamisch erzeugte Dokumente erneut an, werden diese häufig ebenfalls nicht aktualisiert. Das Problem besteht allerdings nur dann, wenn der Benutzer dieselbe IE-Instanz verwendet und die Dokumente über URLs oder über Verweise anfordert. Für Datenbestände, die sich häufig ändern, kann das zum Problem werden. Stellen Sie sich vor, der Chef eines Unternehmens fordert in regelmäßigen Abständen über einen Verweis einer Internetanwendung eine ASP-Seite an, die den aktuellen Umsatz aus den Unternehmensdaten berechnet. Falls er den Browser zwischenzeitlich nicht schließt, wird er immer dieselben Daten erhalten, zumindest so lange, bis das Verfallsdatum des Dokuments erreicht ist.

Für dieses Problem kenne ich zwei Lösungen: Eine einfache Lösung für den Anwender bzw. den Tester ist die Betätigung des Aktualisieren-Schalters (oder von F5). Damit fordert der IE das Dokument tatsächlich neu an, auch wenn dieses noch im Cache zu finden ist. Eine andere Lösung ist, am Beginn des ASP-Dokuments dessen Verfallsdatum auf »Sofort« zu setzen: `Response.Expires = 0`. Dann holt der IE das Dokument nicht aus seinem Cache, sondern ruft dieses immer neu vom Webserver ab.

7.2 Anwendungen und Sitzungen

In ASP-Programmen arbeiten Sie sehr oft mit Anwendungen (die über das `Application`-Objekt repräsentiert werden) und mit Sitzungen (die Sie über das `Session`-Objekt erreichen). Das Wissen um Anwendungen und Sitzungen gehört also zu den Grundlagen und wird deswegen hier beschrieben (obwohl dieses Thema irgendwie hier nicht passt, aber ich wusste nicht, wo sonst ...).

In Kapitel 3 habe ich ja bereits beschrieben, was Webanwendungen prinzipiell sind und wie Sie diese einrichten. Nur wenn Sie einen Webordner als Anwendung konfigurieren, können Sie globale Daten in Ihren ASP-Dokumenten verwenden und die Ereignisse `OnStart` und `OnEnd` des `Application`- und des `Session`-Objekts auswerten (siehe Kapitel 8 und 9).

Anwendungen

Ist ein Webordner als Anwendung konfiguriert, erzeugt der IIS beim ersten Zugriff eines Benutzers auf ein beliebiges Dokument dieses Webordners eine Anwendung, die normalerweise so lange bestehen bleibt, bis der Webserver heruntergefahren wird (eine Anwendung wird daneben auch dann beendet, wenn die Datei GLOBAL.ASA verändert wurde, siehe Kapitel 9). Eine Anwendung ist im Prinzip ein Speicherbereich, in dem der IIS Daten der Anwendung speichert. Über das ASP-Objekt `Application` erhalten Sie Zugriff auf diese Daten. Hier können Sie z. B. eigene Daten anwendungsweit speichern. Im `OnStart`-Ereignis des `Application`-Objekts können Sie auf den Start, im Ereignis `OnEnd` auf das Beenden der Anwendung reagieren. Das `Application`-Objekt wird im nächsten Kapitel, das Speichern von globalen Daten und der Umgang mit den Ereignissen in Kapitel 9 behandelt.

Sitzungen

Sitzungen sind bei der ASP-Programmierung enorm wichtig. Nur über Sitzungen können Sie Daten global, aber für die einzelnen Webbenutzer getrennt speichern. Der IIS (ASP) bzw. das .NET-Framework (ASP.NET) erzeugen immer dann eine neue Sitzung, wenn ein Client eine ASP-Seite einer sitzungsfähigen Anwendung anfordert, für den noch keine Sitzung besteht. Eine Sitzung ähnelt einer Anwendung,

nur dass die Daten der Sitzung separat für jeden Client gespeichert werden. Rufen z. B. zwei Internet-Explorer-Instanzen ein ASP-Dokument ab, erzeugt der IIS zwei Sitzungen. Das ist sogar dann der Fall, wenn beide Instanzen auf demselben Rechner ausgeführt werden.

Wie beim `Application`-Objekt können Sie im `OnStart`-Ereignis des `Session`-Objekts auf den Start, im Ereignis `OnEnd` auf das Ende der Sitzung reagieren, z. B. um beim Start eine Datenbankverbindung zu erzeugen, die beim Beenden wieder geschlossen wird.

Eine Sitzung wird immer dann beendet, wenn der Anwender so lange kein Dokument der Anwendung abgerufen hat, wie in der Einstellung `Sitzungstimeout` der Anwendung (in Minuten) eingestellt ist. Diese Einstellung können Sie in den Eigenschaften des Webordners über die Konfiguration der Anwendung ändern. Alternativ können Sie das Timeout auch in ASP über die Eigenschaft `Timeout` des `Session`-Objekts einstellen. Ruft der Anwender permanent Dokumente ab, bleibt die Sitzung so lange bestehen, bis der Anwender eine Pause macht, die dem Timeout entspricht. Natürlich wird eine Sitzung auch dann beendet, wenn der Webserver heruntergefahren wird. Wird die Sitzung beendet, gibt der IIS alle reservierten Speicherbereiche frei. Damit gehen natürlich auch die im `Session`-Objekt gespeicherten Daten verloren.

Das `Session`-Objekt wird im nächsten Kapitel, das Speichern von globalen Daten und der Umgang mit den Ereignissen in Kapitel 9 behandelt.

7.3 Der Aufbau eines ASP-Dokuments

Ein ASP-Dokument ist ein um eingebettete ASP-Programme erweitertes HTML-Dokument mit der Dateiendung .ASP. An dieser Endung erkennt der Webserver, dass es sich um ein ASP-Dokument handelt. ASP-Dokumente müssen nicht zwangsläufig HTML- oder ASP-Code enthalten. Denkbar sind auch solche Dokumente, die aus reinem ASP-Code bestehen oder die gar keinen ASP-Code enthalten. Wirklich sinnvoll ist allerdings nur die Kombination von HTML und ASP. Ein einfaches ASP-Dokument könnte so aussehen:

```
<%@Language="VBScript"%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
<title>Hello World</title>
</head>
<body>

<h1>Hello World in ASP</h1>

<%
 For i = 1 to 3
    Response.Write "Hello World<br>"
 Next
%>

</body>
</html>
```

Die im Beispiel hervorgehobenen Zeilen sind ASP-Code, hier in der Programmiersprache VBScript. Dieser Code ist in die ASP-Token `<%` und `%>` eingebettet. An diesen Token erkennt die ASP-Engine, dass es sich um ASP-Code handelt, und führt diesen aus.

Die erste Zeile im Beispiel ist eine spezielle ASP-Anweisung, die der ASP-Engine die verwendete Sprache mitteilt. In ASP 2 und 3 können Sie standardmäßig hier die Sprachen *JScript* (Microsofts Adaption von JavaScript) und *VBScript* einsetzen. Wenn Sie weitere Script-Engines auf Ihrem System installiert haben (falls es die überhaupt gibt, ich habe im Internet keine gefunden), können Sie auch die entsprechenden Sprachen einsetzen. In ASP.NET können Sie alle in .NET verfügbaren Sprachen einsetzen, z. B. C# oder VB, wie ich es in Kapitel 11 beschreibe.

Der eigentliche Programmcode erzeugt in einer Schleife mit Hilfe des `Response`-Objekts (das Bestandteil der ASP-Engine ist) die dreimalige Ausgabe von »Hello World«.

Beachten Sie, dass ein ASP-Dokument nicht wie ein HTML-Dokument direkt im Webbrowser geöffnet werden kann. Damit die in einem solchen Dokument enthaltenen Programme auch ausgeführt werden, muss das ASP-Dokument in einem Ordner des Webservers gespeichert sein und über diesen geöffnet werden.

ASP-Dokumente erzeugen immer HTML-Dokumente, die vom Browser ganz normal ausgewertet werden können. Da diese HTML-Dokumente natürlich auch clientseitige Skripte mit DHTML-Zugriff und CSS enthalten können, hängt die erfolgreiche Darstellung wie bei HTML vom verwendeten Browser ab. Wenn Sie in Ihren ASP-Programmen reinen HTML-Code erzeugen (ohne DHTML/JavaScript und CSS), können diese Dokumente in den meisten verfügbaren Browsern betrachtet werden. Aber das wissen Sie ja bereits.

Der Browser erhält in unseren Beispiel den folgenden Quellcode:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
<title>Hello World</title>
</head>
<body>

<h1>Hello World in ASP</h1>

Hello World<br>Hello World<br>Hello World<br>

</body>
</html>
```

Das Ergebnis zeigt Abbildung 7.1.

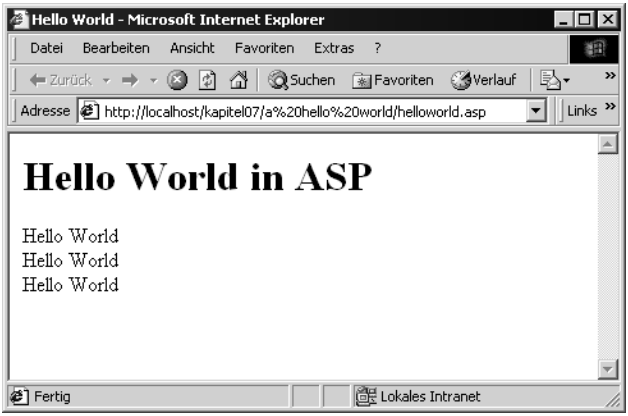


Bild 7.1: Hello World in ASP im Internet Explorer

ASP-Programme mit ASP.NET

Dieses Kapitel behandelt hauptsächlich die klassische ASP-Programmierung mit VBScript, damit die erzeugten Programme auch ohne das .NET-Framework laufen. Wenn Sie dieses auf Ihrem System aber installiert haben, können Sie statt VBScript auch C# oder VB.NET als Programmiersprache verwenden. Der im Moment wichtigste Vorteil wäre, dass diese Sprachen typsicher sind und die Deklaration von Variablen und anderen Daten mit richtigen Datentypen erlauben. Das Buch beschreibt ab Kapitel 11 die Programmierung unter .NET noch ausführlicher. Wenn Sie aber schon die Beispiele dieses und des nächsten Kapitels statt mit VBScript mit VB.NET oder mit C# ausprobieren wollen, müssen Sie Ihre ASP-Dateien ein wenig ändern. Vergeben Sie den Dateien zunächst die Dateiendung .ASPX. Der Webserver sendet diese Dateien dann nicht zu ASP.DLL, sondern an eine ISAPI-Erweiterung, die die Programme über das .NET-Framework ausführt. Die zweite Änderung ist, dass Sie im ASP-Attribut `language` »C#« oder »VB« angeben. Beachten müssen Sie dann noch, dass Sie eigene Funktionen und Prozeduren im `script`-Tag erzeugen müssen und nicht, wie bei ASP, dazu die ASP-Token verwenden können.

Einbinden von ASP-Programmen

ASP-Programme können Sie über die ASP-Token <% und %> an beliebiger Stelle im Dokument einbinden. Innerhalb dieser Token darf Quellcode stehen, der der normalen Syntax der verwendeten Scriptsprache entspricht. Das können normale Anweisungen, aber auch Funktionen sein. Enthält ein ASP-Block Anweisungen, die nicht zu einer Funktion oder Prozedur gehören, werden diese Anweisungen bei der Auswertung des Dokuments an der Stelle ausgeführt, an der sie stehen. Anweisungen in Funktionen oder Prozeduren werden erst dann ausgeführt, wenn ein andere Teil des ASP-Dokuments diese Funktion oder Prozedur aufruft. Das folgende Beispiel zeigt dies an Hand einer einfachen Prozedur, die im Head-Bereich des Dokuments deklariert ist und im Body-Bereich aufgerufen wird:

```
<%@Language="VBScript"%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
<title>Hello World</title>

<%
 ' Diese Prozedur wird erst dann ausgeführt,
 ' wenn sie explizit aufgerufen wird
 Sub SayHello()
    Response.Write "Hello World"
 End Sub
%>

</head>

<body>

<%
 For i = 1 to 3
   'Aufruf der Prozedur
   SayHello
```

```
 Next
%>

</body>
</html>
```

Die ASP-Engine arbeitet ein ASP-Dokument von oben nach unten sequenziell ab. HTML-Quellcode wird dabei ohne weitere Verarbeitung in das Ergebnisdokument übernommen. Programme in ASP-Token werden an der Stelle, an der sie im Quelltext auftreten, ausgeführt, wenn es sich nicht um Funktionen oder Prozeduren handelt. Die ASP-Engine übernimmt die Ausgabe dieser Programme dann an der Stelle, an der die Programme stehen, ebenfalls in das Ergebnisdokument. ASP ersetzt quasi den ursprünglichen Inhalt der ASP-Token durch die Ausgabe des Programms.

Soll innerhalb von ASP-Token HTML-Quellcode erzeugt werden, kann dies über die `Write`-Methode des `Response`-Objekts geschehen:

```
<%Response.Write "Hello World"%>
```

Alternativ können Sie zum einfachen Schreiben auch eine spezielle Syntax verwenden. Hierbei stellen Sie den Zuweisungsoperator der verwendeten Programmiersprache an den Anfang des Quelltextes und programmieren rechts davon einen Ausdruck:

```
<%="Hello World"%>
```

Diese zweite Version kann nur für einfache Ausdrücke verwendet werden. Das folgende Beispiel demonstriert dies an Hand der Ausgabe des aktuellen Datums und der aktuellen Zeit:

```
<p>Heute ist der <%=Date%></p>
<p>Hier ist gerade <%=Time%> Uhr</p>
```


Auf die Variante des Schreibens von HTML-Quellcode mit »= Ausdruck« sollten Sie verzichten. Die ASP-Engine macht aus »= Ausdruck« sowieso »`Response.Write` *Ausdruck«. Wenn Sie* `Response.Write` *direkt verwenden, ist Ihr Quellcode einfacher zu verstehen.*

Eine andere Möglichkeit, ASP-Programme zu erzeugen, ist, den `script`-Tag zu verwenden, wie Sie es bereits von der clientseitigen

Programmierung her gewohnt sind. Sie müssen allerdings der ASP-Engine mit dem Attribut `runat="server"` mitteilen, dass das Script auf dem Server ausgeführt werden soll. Diese Art der Einbindung von Programmen können Sie nur für die Programmierung von Funktionen oder Prozeduren verwenden:

```
<%@Language="VBScript"%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
<title>Hello World</title>

<script language="VBScript" runat="Server">
 Sub SayHello()
    Response.Write "Hello World<br>"
 End Sub
</script>

</head>

<body>

<%
 For i = 1 to 3
   'Aufruf der Prozedur
   SayHello
 Next
%>

</body>
</html>
```

Wenn Sie in einem `script`*-Tag Anweisungen unterbringen, die nicht zu einer Prozedur oder Funktion gehören, werden diese erst ausgeführt, nachdem das restliche Dokument komplett ausgewertet wurde.*

Wenn Sie einen `script`-Tag also mitten im Dokument wie ASP-Token verwenden und versuchen, Quelltext zu schreiben, wird dieser Quelltext erst am Ende des Dokuments (hinter `</html>`) eingefügt. Aus dem folgenden Quellcode:

```
<%@Language="VBScript"%>

<html>
<body>

<h1>Hello World-Versuch im script-Tag</h1>

<script language="VBScript" runat="server">
 for i = 1 to 3
    Response.Write "Hello World<br>"
 Next
</script>

</body>
</html>
```

resultiert dieser HTML-Code:

```
<html>
<body>

<h1>Hello World-Versuch im script-Tag</h1>


</body>
</html>
Hello World<br>Hello World<br>Hello World<br>
```

Sie sehen, dass der `script`-Tag zwar ausgewertet wurde, aber erst nachdem das restliche Dokument abgearbeitet wurde. Die Ausgabe der enthaltenen Anweisungen steht damit hinter dem abschließenden `</html>`.

Im klassischen ASP sehe ich nicht allzu viele Vorteile in der Verwendung eines `script`*-Tags für Prozeduren und Funktionen. Der einzige Vorteil wäre, dass Sie in einem* `script`*-Tag auch mit einer anderen Sprache arbeiten können als im übrigen Dokument. In ASP.NET können Sie Prozeduren und Funktionen, die Sie direkt im ASP-Dokument unterbringen wollen (was aber in ASP.NET eigentlich nicht sinnvoll ist, siehe in Kapitel 11), allerdings nicht in ASP-Token einbinden und müssen den* `script`*-Tag verwenden.*

Mischen von ASP und HTML

Sie können ASP und HTML fast beliebig mischen. Das Hello-World-Beispiel kann z. B. auch so programmiert werden:

```
<%
 for i = 1 to 3
    %>
    Hello World<br>
    <%
 Next
%>
```

Obwohl der Text »Hello World
« nun reiner HTML-Code ist, wird er dennoch in der Schleife ausgeführt. Weil diese Technik auch mit `If`-Verzweigungen und dem `Select-Case` funktioniert, macht es manchmal Sinn, ASP mit HTML zu mischen. Beachten Sie dabei aber den Tipp im nächsten Abschnitt.

Tipps und Tricks

Performance erhöhen

Um die Performance eines ASP-Dokuments nicht durch falsche Programmierung zu vermindern, sollten Sie ASP-Code nicht »wild« mit HTML-Code mischen. Bei der Abarbeitung solchen Codes muss die ASP-Engine andauernd zwischen HTML und ASP umschalten. Und das kostet eben Zeit. Der folgende Quellcode zeigt ein solches Negativ-Beispiel:

```
<body>
<%
 Dim i
 For i = 1 To 3
%>
   <h<%=i%>1>Hello World</h<%=i%>>
<%
 Next
%>
...
```

Abgesehen davon, dass dieser Quellcode absolut unübersichtlich ist: Er funktioniert, aber eben langsam.

Die @-Anweisungen

Mit den fünf @-Anweisungen können Sie einige Einstellunge für eine ASP-Seite vornehmen.

Anweisung	Beschreibung
`@Codepage`	Mit dieser Anweisung können Sie den Zeichensatz bestimmen, der vom ASP-Programm verwendet werden soll. Geben Sie dazu die Nummer des Zeichensatzes an. 1252 steht für den in Europa und USA verwendeten Zeichensatz. Der hier angegebene Zeichensatz gilt allerdings nur für den HTML-Code, der von ASP-Anweisungen erzeugt wird. Für den normalen HTML-Code der Seite gilt der in Kapitel 4 beschriebene Meta-Tag. Die Voreinstellung ist der Standard-Zeichensatz des Webservers.
`@EnableSessionState`	Wenn Sie `@EnableSessionState` auf `False` setzen, verwaltet die ASP-Engine für diese Seite keinen Sitzungsstatus. Die Ausführung der Seite wird damit beschleunigt. Sie sollten für Seiten, die keinen Sitzungsstatus benötigen, `@EnableSessionState` immer auf `False` setzen. Die Voreinstellung ist die, die in der Konfiguration des Webordners angegeben ist.

Anweisung	Beschreibung
`@Language`	Wie Sie ja bereits wissen, geben Sie hier die Sprache an, mit der Sie innerhalb der ASP-Dokumente arbeiten.
`@LCID`	Mit dieser Anweisung können Sie das Gebietsschema für die Seite festlegen. Das Gebietsschema bestimmt, wie Zahl-, Währungs-, und Datumswerte formatiert werden, wenn Sie diese Werte mit entsprechenden Funktionen in ASP ausgeben. Geben Sie 1031 für das deutsche Schema und 1033 für das Schema der USA an. Voreinstellung ist das Standard-Schema des Servers.
`@Transaction`	Mit dieser Anweisung bestimmen Sie das Transaktionsverhalten der Seite. Transaktionen werden in diesem Buch nicht behandelt. Sie finden auf der Website einen Artikel zu Transaktionen mit COM+ und dem MTS.

Tabelle 7.1: Die @-Anweisungen von ASP

7.4 Debuggen von ASP-Programmen

Ich beschreibe das Debuggen von ASP-Anwendungen bereits hier, bevor Sie überhaupt mit ASP beginnen zu programmieren. Das hat zwei Gründe: Einmal machen Sie wahrscheinlich (genau wie ich) am Anfang die meisten Fehler, die dann natürlich beseitigt werden müssen. Da ASP Programme nicht kompiliert sondern interpretiert werden, werden sogar Syntaxfehler als Laufzeitfehler gemeldet. Der andere Grund ist, dass Sie unter Windows 2000 u. U. (je nach Service Pack) mit einer eigenartigen Fehlermeldung konfrontiert werden, die absolut nichts mit Ihrem eigentlichen Fehler zu tun hat.

Beim Debuggen können Sie unter drei Möglichkeiten wählen:

- Sie verwenden beim Auftreten eines Laufzeitfehlers einfach die integrierte ASP-Fehlermeldung, die sogar die Zeile angibt, in der der Fehler aufgetreten ist.

- Sie schalten das serverseitige Scriptdebuggen ein und verwenden den Debugger.
- Sie geben Ausdrücke und Inhalte von Variablen mit `Response.Write` aus.

Die einfachste Debugging-Technik ist, die ASP-Ausgabe zu verwenden, die bei einem Laufzeitfehler erzeugt wird. Der IIS wertet aufgetretene Laufzeitfehler (die dem HTTP-Fehler 500 entsprechen) über eine ASP-Datei 500-100.ASP aus, wenn das Scriptdebuggen für eine Webanwendung nicht eingeschaltet ist (was per Voreinstellung der Fall ist). Die Anzeige eines Laufzeitfehlers sieht dann etwa aus wie in Abbildung 7.3. Die für uns wichtige Information finden Sie im unteren Bereich der Abbildung.

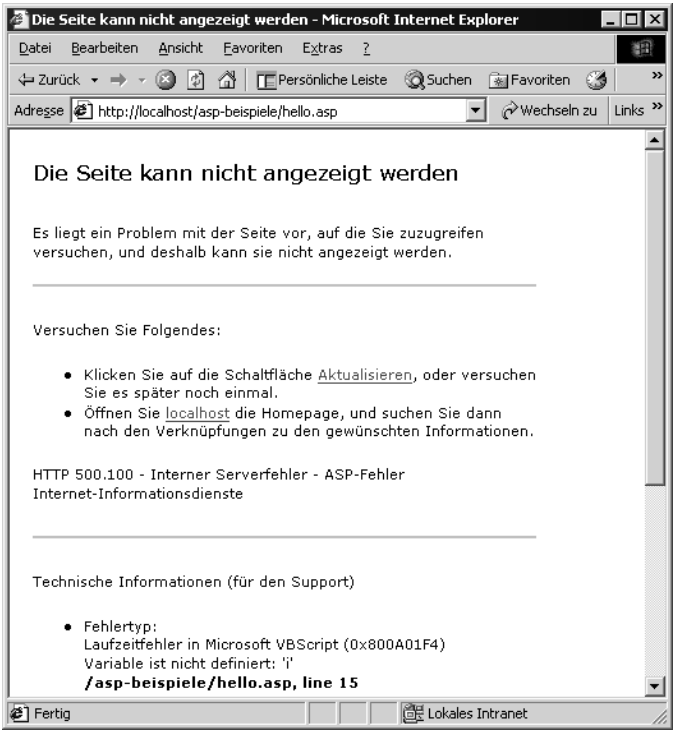


Bild 7.2: Anzeige eines Laufzeitfehlers in ASP

Wundern Sie sich nicht, wenn es bei Ihnen anders aussieht. Die ASP-Datei 500-100.ASP *besitzt im aktuellen Windows 2000 SP 2 nämlich selbst einen Fehler. Falls das auf Ihrem System auch der Fall ist, zeigt ASP fälschlicherweise einen Kompilierungsfehler in dieser Datei an Zeile 129 an. Beseitigen Sie dann einfach den Fehler in der Datei* 500-100.ASP*, die Sie normalerweise im Ordner* C:\WINNT\HELP\IISHELP\COMMON *finden. In der Zeile 127 müssen Sie das* `Response.Write` *dazu eine Zeile tiefer stellen.*

Bei Syntaxfehlern wie dem in Abbildung 7.3, bei dem ich einfach vergessen habe, eine Variable zu deklarieren, ist das einfache Debuggen noch recht gut zu verwenden. Bei logischen Fehlern können Sie damit aber natürlich nichts anfangen, weil diese meist keinen Laufzeitfehler erzeugen. Um diese Fehler debuggen zu können, benötigen Sie zunächst einen Debugger für Script-Programme. Wenn Sie nur den IIS installiert haben, ist normalerweise der Microsoft Script Debugger gleich mit installiert worden. Dieser einfache Debugger ist schon recht hilfreich. Haben Sie Visual Interdev installiert, ersetzt der bessere integrierte Debugger von Visual Interdev den Microsoft Script Debugger. Falls Sie Visual Studio.NET installiert haben, besitzen Sie einen weiteren Debugger (der allerdings parallel zu vorhandenen Debuggern verwendet werden kann).

Grundlage des Debuggens ist, dass der Webordner als Anwendung konfiguriert und das Debuggen für diese Anwendung freigeschaltet ist. Schalten Sie die Option ASP-SERVERSEITIGES SCRIPTDEBUGGEN AKTIVIEREN für den Webordner in der Konfiguration der Anwendung ein, dann hält der Debugger bei Laufzeitfehlern an und Sie können debuggen.

Diese Option sollten Sie nie auf einem Rechner einschalten, der für geschäftliche Zwecke eingesetzt wird. Tritt bei eingeschalteter Option ein Laufzeitfehler auf, hält der Debugger den Prozess, in dem die betroffene ASP-Seite ausgeführt wird (je nach Isolation ist das der IIS-Prozess selbst oder der DLLHOST.EXE*-Prozess), so lange an, bis jemand das Debuggen von Hand beendet. Bis dahin kann kein Benutzer mehr Dokumente von diesem Webordner oder vom gesamten Web abrufen.*

Wenn Sie dies einmal ausprobieren wollen, beachten Sie, dass der Internet Explorer Webseiten in seinem Cache zwischenspeichert. Wenn Sie eine schon einmal zuvor geladene HTML- oder ASP-Seite noch einmal anfordern, sieht es oft nur so aus, als ob diese vom Webserver gesendet wird. In Wirklichkeit wird die Seite dann aus dem Cache gelesen.

Ist der Webordner zum Debuggen freigeschaltet, startet der Debugger automatisch beim Eintritt eines Laufzeitfehlers, wenn nur ein Debugger installiert ist. Haben Sie z. B. neben dem Script Debugger auch Visual Studio.NET installiert, erhalten Sie zunächst die Möglichkeit, den zu verwendenden Debugger auszuwählen. Da ich Visual Studio.NET in Kapitel 12 beschreibe, gehe ich hier davon aus, dass Sie den Scriptdebugger verwenden.

Der Scriptdebugger

Der Scriptdebugger hält den IIS- bzw. DLLHOST-Prozess beim Eintritt eines Laufzeitfehlers automatisch an und zeigt den Quellcode der ASP-Seite an. Er hält dabei an der Zeile an, die der Zeile folgt. Ein ASP-Dokument mit dem folgenden Quellcode

```
<%
Dim steuer, gehalt, steuersatz
gehalt = 10000
steuer = gehalt / steuersatz * 100
Response.Write steuer
%>
```

erzeugt z. B. einen Laufzeitfehler »Division durch Null« weil die Variable *steuersatz* nicht initialisiert wurde.

Wenn Sie den Dialog bestätigen, der den Fehler anzeigt, können Sie den Fehler im Scriptdebugger analysieren. Der Debugger zeigt die Zeile, in der der Fehler aufgetreten ist, mit einem Pfeil an und hat den Programmcode an dieser Zeile angehalten. Sie können nun das Befehlsfenster (»Command Window«) (VIEW | COMMAND WINDOW) verwenden, um den Inhalt von Variablen oder auch komplexe Ausdrücke auszugeben. Verwenden Sie dazu die Syntax `?` *Ausdruck*. Das Fragezeichen ist dabei eine Kurzform der `Print`-Anweisung, die beliebige Ausdrücke im Befehlsfenster ausgeben kann.

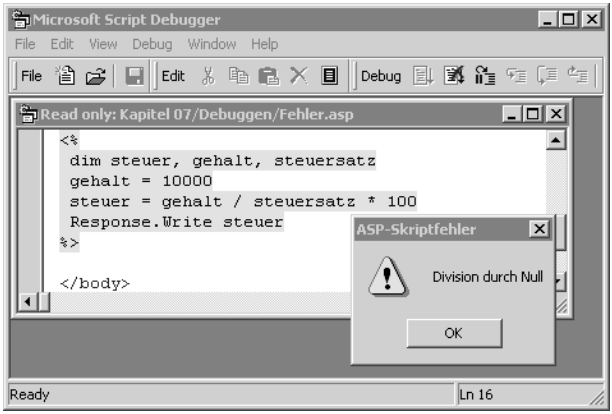


Bild 7.3: Der Scriptdebugger hält bei einem Laufzeitfehler an

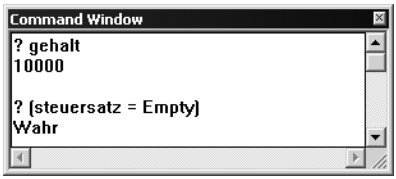


Bild 7.4: Ausgabe einer Variablen und eines Ausdrucks im Befehlsfenster des Scriptdebuggers

Den angehaltenen Programmcode können Sie mit F8 schrittweise weiter ausführen (was eigentlich nur bei logischen Fehlern sinnvoll ist). Der Debugger springt dann auch in aufgerufene Prozeduren oder Funktionen hinein und lässt Sie diese schrittweise durchlaufen. Wenn Sie nicht in Prozeduren verzweigen wollen, betätigen Sie ⇧ + F8. Mit F5 können Sie das Programm ab der aktuellen Zeile zu Ende ausführen. Das sollten Sie immer auch machen, da der IIS ansonsten blockiert ist.

Leider lässt ein Scriptdebugger prinzipiell nicht zu, dass der Programmcode während des Debuggens verändert wird. Sie müssen sich also darauf beschränken, den Fehler zu finden, das Programm mit F5 zu Ende auszuführen und den Fehler mit einem Editor in der ASP-Datei zu beseitigen.

Debuggen von logischen Fehlern

Bei logischen Fehlern hält der Debugger meist nicht automatisch an, weil diese nur selten einen Laufzeitfehler erzeugen. Um logische Fehler debuggen zu können, müssen Sie einen Haltepunkt setzen. Ein Haltepunkt bewirkt, dass der Debugger an der Stelle anhält. Sie können Ihren Programmcode dann wie bei einem Laufzeitfehler debuggen. Haltepunkte können Sie mit zwei Techniken setzen: Die einfachste Technik ist das Anbringen der Anweisung `Stop` im Programmcode. Der Debugger hält das Programm automatisch an einer solchen Anweisung an. `Stop`-Anweisungen sollten Sie wieder aus dem Programmcode entfernen, obwohl die Script-Engine (in der Version 5.5) diese Anweisungen ignoriert, wenn das Debuggen nicht eingeschaltet ist.

Eine andere, etwas kompliziertere Möglichkeit ist, im Scriptdebugger einen Haltepunkt zu setzen. Dazu müssen Sie die ASP-Datei in Scriptdebugger öffnen, den Sie im Startmenü unter PROGRAMME | ZUBEHÖR | MICROSOFT SCRIPT DEBUGGER finden. Ein einfaches Öffnen als Datei funktioniert aber nicht, weil der Debugger sich in den ausführenden Prozess einhängen muss, um Haltepunkte zu ermöglichen. Gehen Sie idealerweise folgendermaßen vor: Öffnen Sie die fehlerhafte ASP-Datei im Browser (natürlich über den Webserver). Öffnen Sie das »Running Documents«-Fenster über das `View`-Menü im Debugger. Suchen Sie dort das fehlerhafte Dokument unter dem Eintrag Standardwebsite (nicht unter dem Browsernamen, die Einträge dort werden für das Debuggen von clientseitigen Scripts verwendet) und öffnen Sie dieses über einen Doppelklick.

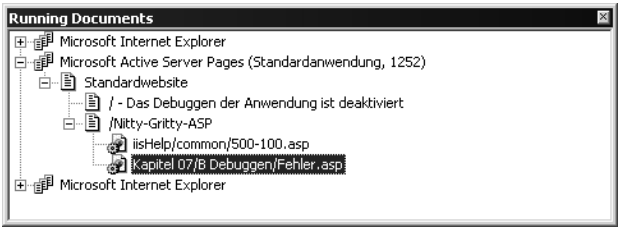


Bild 7.5: Das Fenster mit den laufenden Dokumenten im Script Debugger

Der Eintrag »Das Debuggen der Anwendung ist deaktiviert« in Abbildung 7.9 bezieht sich übrigens auf den Stammordner meines Webservers.

Im geöffneten ASP-Dokument können Sie nun mit F9 einen Haltepunkt setzen und auch wieder entfernen. Aktualisieren Sie das Dokument im Browser, hält der Debugger die Ausführung am Haltepunkt an.

Irgendwie ist die `Stop`-Anweisung einfacher, oder?

7.5 VBScript in ASP

ASP-Programme können in jeder beliebigen Scriptsprache geschrieben werden. Voraussetzung dafür ist, dass auf dem Server eine passende Script-Engine installiert ist. Mit dem IIS wird eine Script-Engine für VBScript und eine für JScript installiert. So könnten Sie Ihre ASP-Programme also in einer dieser Sprachen entwickeln.

VBScript hat sich aber für ASP durchgesetzt. Das liegt wahrscheinlich daran, dass diese Scriptsprache näher an den verschiedenen Microsoft-Technologien angesiedelt ist als z. B. JScript. Ich beschreibe also in diesem Buch ausschließlich VBScript zur Verwendung in ASP. Sie können die Beispiele jedoch auch ohne viel Probleme in einer anderen Sprache umsetzen, da ASP-Programme in allen Sprachen prinzipiell identisch sind. Auf eventuelle Unterschiede weise ich im Einzelfall hin.

Referenzen

Die Microsoft VBScript-Referenz finden Sie leider nicht in der Hilfe zum IIS (5), sondern im Internet. Die Referenz der jeweils aktuellen Version finden Sie über die Seite *msdn.microsoft.com/scripting/vbscript/default.htm*. Eine deutsche Version der Referenz habe ich bei Microsoft leider nicht gefunden (der Link in der IIS-Hilfe ist leider tot)!? Eine recht gute Referenz finden Sie auch bei DevGuru: *www.devguru.com/Technologies/vbscript/quickref/vbscript_intro.html*. Auf der Buchseite bei *www.nitty-gritty.de* finden Sie zudem einen umfangreicheren Artikel zu VBScript. Die Funktionen werden in diesem Artikel kategorisiert und umfangreich beschrieben.

Die VBScript-Version

VBScript existiert mittlerweile in den Versionen 1.0, 2.0, 3.0, 4.0, 5.0, 5.5 und 5.6. Jede Version bringt einige Erweiterungen mit sich. Ab Version 5.0 können Sie in VBScript z. B. eigene Klassen deklarieren. Viele in ASP verwendbare Features hängen also stark von der auf dem Server installierten Script-Version ab. Der IIS 4 installiert per Default die Version 3.0, der IIS 5 die Version 5.0 und das .NET-Framework die Version 5.6. Wenn Sie noch mit dem IIS 4 arbeiten, sollten Sie die Script-Engine auf die aktuelle Version aktualisieren. Dann können Sie in ASP auch die neueren VBScript-Features nutzen. Sie finden die aktuelle Version unter *http://msdn.microsoft.com/scripting*.

Features, die nicht unter VBScript 3.0 verfügbar sind, habe ich im Buch übrigens mit der jeweils gültigen Version gekennzeichnet.

7.5.1 Bezeichner

Wie in anderen Sprachen auch, dürften Bezeichner für Variablen, Konstanten, Funktions- und Prozedurnamen, Argumentnamen, Klassennamen etc. in VBScript nur aus Buchstaben, Zahlen und dem Unterstrich bestehen. Das erste Zeichen eines Bezeichners darf nur ein Buchstabe sein. Ein gültiger Bezeichner ist z. B. »Frage_Auf_Antwort_42«. Ungültige Bezeichner resultieren in ASP in dem Laufzeitfehler »Ungültiges Zeichen« (wenn der Bezeichner mit einem Unterstrich beginnt) oder »Bezeichner erwartet« (wenn der Bezeichner mit einer Zahl beginnt). Die Länge eines Bezeichners darf 255 nicht überschreiten.

7.5.2 Anweisungen, Ausdrücke und Operatoren

Anweisungen

In VBScript werden Anweisungen nicht, wie in anderen Sprachen, abgeschlossen. Das Ende einer Anweisung ist durch das Ende der geschriebenen Zeile gekennzeichnet. Daraus folgt, dass eine Anweisung nicht einfach umbrochen werden kann und Sie in einer Zeile nicht direkt mehrere Anweisungen schreiben dürfen. Um Anweisungen in mehrere Zeilen zu umbrechen, müssen Sie einen Unterstrich an das Ende der Zeile hängen. Die Anweisung

```
bruttopreis = nettopreis * 1.16
```

können Sie so umbrechen:

```
bruttopreis = _
   nettopreis * 1.16
```

Wollen Sie eine Zeile mitten in einer Textkonstante umbrechen, können Sie nicht einfach den Unterstrich ansetzen.

```
Response.Write "Das ist ein nicht funktionierender _
   Versuch mit dem Umbruch einer Textkonstante"
```

Bei Textkonstanten müssen Sie dem Compiler schon mitteilen, wo die Textkonstante aufhört und wieder anfängt. Addieren Sie dazu einzelne Textkonstanten mit dem Textadditions-Operator (&):

```
Response.Write "Das ist ein funktionierender Versuch " & _
   "mit dem Umbruch einer Textkonstante"
```

Wollen Sie mehrere Anweisungen in einer Zeile unterbringen, hängen Sie einen Doppelpunkt an die Zeile an:

```
i = i + 1: Response.Write i
```

Mehrere Anweisungen in einer Zeile machen den Quellcode unübersichtlich. Schreiben Sie also grundsätzlich (mit wenigen Ausnahmen) nur eine Anweisung pro Zeile.

Ansonsten bauen sich Anweisungen auf wie in anderen Programmiersprachen prinzipiell auch. Zuweisungen verwenden den Zuweisungsoperator = (der in VBScript identisch ist mit dem Vergleichsoperator).

Kommentare

In VBScript leiten Sie einen Kommentar durch ein einfaches Apostroph ein:

```
Response.Write "Hello World" ' Das ist ein Kommentar
```

Alles, was dem Kommentarzeichen folgt, wird von der Script-Engine nicht als Programmcode gewertet. Mehrzeilige Kommentare sind in VBScript nur möglich, indem das Kommentarzeichen jeweils vor die Zeile geschrieben wird:

```
' Das ist ein
' mehrzeiliger
' Kommentar
```

Statt des Apostrophs können Sie theoretisch auch das Schlüsselwort `Rem` verwenden, aber in diesem Fall werden Sie von der Programmierergemeinde gesteinigt ...

Der Aufruf von Funktionen, Prozeduren und Methoden

Beim Aufruf von Prozeduren, oder wenn Sie Funktionen so aufrufen wollen, dass der Rückgabewert nicht verwendet wird, konfrontiert Sie VBScript mit zwei Syntaxvarianten. Sie können dazu das Schlüsselwort `Call` verwenden:.

```
Call Response.Write("Hello World")
```

Alternativ können Sie `Call` auch weglassen, dann dürfen Sie allerdings nicht klammern:

```
Response.Write "Hello World"
```

Häufig machen VBScript-Anfänger den Fehler, `Call` wegzulassen und trotzdem zu klammern:

```
Response.Write ("Hello World")
```

Wenn nur ein Argument vorhanden ist, funktioniert diese Variante prinzipiell, bei mehreren Argumenten resultiert der Versuch die Argumente zu klammern in einem Syntaxfehler. Diese Variante hat aber auch einen erheblichen Nachteil: Argumente, die mit »Call By Reference« übergeben werden (siehe Seite 181), werden damit explizit mit »Call By Value« übergeben, wenn diese geklammert werden. Das kann zu sehr schwer lokalisierbaren logischen Fehlern im Programm führen. Merken Sie sich deshalb:

Wenn die Argumente einer Funktion oder Prozedur geklammert werden, muss links vom Namen der Funktion bzw. Prozedur etwas stehen (außer dem Doppelpunkt natürlich). Wenn es sich nicht um eine Zuweisung handelt, kann das Schlüsselwort `Call` *verwendet werden.*

Funktionen werden wie in anderen Sprachen aufgerufen, wenn Sie den Rückgabewert verwenden wollen. Die `Round`-Funktion, die eine Zahl rundet (hier auf zwei Nachkommastellen), rufen Sie so auf:

```
ergebnis = Round(zahl, 2)
```

Viele Prozeduren, Funktionen und Methoden besitzen optionale Argumente, die Sie in der Syntaxbeschreibung an den eckigen Klammern erkennen. Die Syntax der `Replace`-Funktion ist ein gutes Beispiel dafür:

```
Replace(expression, find, replacewith _
   [, start[, count[, compare]]])
```

Die optionalen Argumente können Sie einfach weglassen.

Was die `Replace`-Syntax bereits andeutet, ist die Tatsache, dass Sie optionale Argumente in VBScript dann angeben müssen, wenn Sie ein folgendes optionales Argument angeben wollen. Sie können also nicht

```
Response.Write Replace(name, "x", "X", , 1)
```

schreiben, um das start-Argument auszulassen und das count-Argument anzugeben. Die Script-Engine meldet dann den Fehler »Typen unverträglich«. Geben Sie stattdessen alle vorausgehenden optionalen Argumente an. Im folgenden Beispiel wird einfach für `start` `1` angegeben, damit die Ersetzung ab dem ersten Zeichen vorgenommen wird:

```
Response.Write Replace(name, "x", "X", 1, 1)
```

Arithmetische Operatoren

VBScript verwendet die arithmetischen Operatoren `()`, `^` (Potenzierung), `*`, `/`, `\` (Integer-Division), `Mod` (Restwert-Division), `+`, `-` und `&` (String-Addition). Wenn Sie Strings addieren wollen, sollten Sie dazu nicht den +-Operator verwenden, sondern `&`. In seltenen Fällen, bei der Verknüpfung von Strings mit Variablen, die `Null` (VBScript) bzw. `DBNull` (VB.NET) speichern, resultiert ansonsten oft ein Laufzeitfehler »Ungültige Verwendung von Null«, wenn der Ergebnis-String weiter verwendet werden soll. Zumindest ist aber der Ergebnis-String leer.

Dieses Problem tritt meist bei der Abfrage von Datenbank-Tabellen auf, bei denen einzelne Felder auch leer sein können. Verknüpfen Sie mit `&`, wird `Null` bzw. `DBNull` automatisch in einen Leerstring konvertiert und das Ergebnis ist in Ordnung.

Vergleichsoperatoren

Für Vergleiche verwendet VBScript die Operatoren `=`, `<`, `<=`, `>`, `>=` , `<>` (ungleich) und `Is` (Vergleich zweier Objekte auf dieselbe Klasse). Stringvergleiche werden wie üblich nach dem ANSI-Code vorgenommen, wobei das erste Zeichen die größte Priorität besitzt (der String »100« ist z. B. *kleiner* als der String »20«). Beim Vergleich von Strings unterscheidet VBScript standardmäßig Groß- und Kleinschreibung. Der Ausdruck `"a" > "A"` ist also zunächst wahr. Wenn Sie beim Vergleich Groß- und Kleinschreibung ignorieren wollen, müssen Sie die `StrComp`-Funktion in der folgenden Form verwenden:

StrComp(*string1*, *string2*, vbTextCompare)

Ist *string1* kleiner als *string2,* gibt diese Funktion -1 zurück. Sind beide Strings gleich, ist die Rückgabe 0. Wenn *string1* größer ist als *string2*, wird 1 zurückgegeben.

Die Möglichkeit, den Textvergleich über die Visual Basic-Anweisung `Option Compare Text` *zu beeinflussen, besteht in ASP nicht.*

Logische und bitweise Operationen

Die Operatoren für logische und bitweise Operationen sind in VBScript identisch. VBScript erkennt meist aus dem Kontext, ob eine logische oder eine bitweise Operation verwendet werden soll. Wenn diese automatische Erkennung jedoch fehlschlägt, müssen Sie mit Klammern nachhelfen.

Operator	Bedeutung für Ausdrücke	Bedeutung für nummerische Werte
`Not`	Negiert einen Ausdruck	Kippt alle Bits um (aus 00001111 wird z. B. 11110000)
`And`	*Ausdruck1* `And` *Ausdruck2* ergibt `True`, wenn beide Ausdrücke `True` ergeben. Ergibt einer der Ausdrücke `Null`, so ergibt `And` ebenfalls `Null`.	*Num1* `And` *Num2*: Im Ergebnis ist Bit *n* gesetzt, wenn Bit *n* in beiden nummerischen Werten ebenfalls 1 ist (0101 `And` 0100 ergibt z. B. 0100).
`Or`	*Ausdruck1* `Or` *Ausdruck2* ergibt `True`, wenn einer der beiden Ausdrücke `True` ergibt. Ist ein Ausdruck `Null` und der andere `False`, ist das Ergebnis `Null`, in allen anderen Fällen `False`.	*Num1* `Or` *Num2*: Im Ergebnis ist Bit *n* gesetzt, wenn Bit *n* in einem der beiden nummerischen Werte ebenfalls 1 ist (0101 `Or` 1001 ergibt z. B. 1101).

Tabelle 7.2: Die wichtigsten logischen und bitweisen Operatoren von VBScript

Beachten Sie die Priorität der logischen Operatoren: `Not` *wird vor* `And` *ausgewertet und* `And` *vor* `Or`*. Zur Sicherheit sollten Sie kombinierte logische Operationen immer klammern. Besonders, wenn ein Operator auch als bitweiser Operator ausgewertet werden könnte, sollten Sie klammern, um Fehler zu vermeiden.*

7.5.3 Datentypen und Variablen

Datentypen

VBScript kennt leider (wie JavaScript) nur den Datentyp `Variant`. Dieser Datentyp kann prinzipiell alles speichern, egal ob es sich dabei um Zahlen, Strings, Arrays oder sogar Objekte handelt. Eine weitere Besonderheit ist, dass ein Variant den Wert `Empty` speichert, wenn der noch nie initialisiert wurde:

```
Dim i ' hier speichert i den Wert Empty
i = 10 ' hier speichert i einen Integer-Wert
```

Die Script-Engine konvertiert die Subtypen von Variant-Datentypen normalerweise automatisch in einen passenden Datentyp, wenn dies erforderlich ist. Speichern Sie in einer Variablen zum Beispiel einen String, der eine Zahl beinhaltet, und führen damit eine mathematische Berechnung aus, konvertiert die Script-Engine den String automatisch in eine Zahl:

```
Dim steuersatz
steuersatz = "0,16"
Response.Write 100 * steuersatz
```

Kann die Konvertierung nicht ausgeführt werden, weil ein String zum Beispiel keine gültige Zahl enthält, resultiert ein Laufzeitfehler. Bei der Konvertierung von Strings in Zahlen müssen Sie beachten, dass die Script-Engine die länderspezifischen Einstellungen des Servers verwendet. In Deutschland muss die Zahl 0.16 z. B. in einem String als "0,16" angegeben werden, in den USA allerdings als "0.16".

Mit Hilfe einiger Konvertierungsfunktionen können Sie Varianten auch explizit konvertieren. `CDbl` wandelt einen Varianten z. B. in einen Double-Wert um.

Da Sie bei Berechnungen mit Varianten nie genau wissen, welchen Datentyp die Script-Engine tatsächlich für die Berechnung verwendet, sollten Sie immer konvertieren. Eine Addition von zwei Variablen kann z. B. als Stringaddition ausgeführt werden, wenn die Script-Engine gerade einen schlechten Tag hat (nein, natürlich dann, wenn die Variablen Zeichenketten speichern, was Sie aber nie genau wissen). 1 + 1 ist dann auf einmal 11.

Mit Hilfe der `VarType`-Funktion können Sie den tatsächlichen Datentyp eines Varianten ermitteln. Mit der `IsNumeric`-Funktion können Sie herausfinden, ob ein Variant grundsätzlich nummerische Daten speichert. Mit `IsDate` überprüfen Sie einen Varianten auf ein gültiges Datum.

Implizite und explizite Deklaration

In VBScript können Sie, wie in JavaScript, standardmäßig Variablen einfach verwenden, ohne diese zuvor zu deklarieren:

```
i = 10
Response.Write i
```

Die Variable wird bei deren erster Verwendung automatisch (implizit) deklariert. Da die implizite Variablendeklaration oft zu logischen Fehlern im Programm führt (nämlich dann, wenn Sie aus Versehen den Namen einer Variablen falsch schreiben), können Sie diese auch abschalten. Bauen Sie dazu einfach die Anweisung `Option Explicit` ganz oben im ASP-Dokument ein. Dann sind Sie prinzipiell dazu gezwungen, Variablen explizit zu deklarieren, was normalerweise über das Schlüsselwort `Dim` geschieht:

<%Option Explicit%>

```
<html>
<head><title></title></head>
<body>
<%
 Dim steuersatz
 steuersatz = 0.16
 Response.Write 100 * steiersatz
%>
</body>
</html>
```

Ein Schreibfehler wie im Beispiel (*steiersatz*) führt dann nicht dazu, dass die Script-Engine einfach eine neue Variable erzeugt. Stattdessen erzeugt die Script-Engine einen Laufzeitfehler, der zumindest im Browser angezeigt wird (sofern Sie keine Laufzeitfehlerbehandlung im ASP-Dokument vorgesehen haben).

Solche Laufzeitfehler können Sie natürlich, wie alle anderen Laufzeitfehler auch, debuggen. Auch wenn die Ausgabe des Laufzeitfehlers im Browser nicht allzu schön ist, haben Sie dennoch den großen Vorteil, dass falsch geschriebene Variablennamen nun nicht mehr zu (eventuell schwerwiegenden) logischen Fehlern führen können.

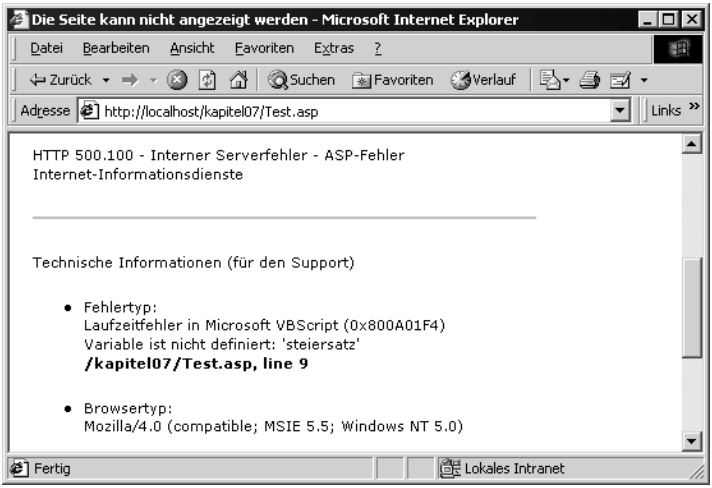


Bild 7.6: Ein Laufzeitfehler, der durch einen falsch geschriebenen Variablennamen erzeugt wurde, wird im Internet Explorer angezeigt

Die Deklaration

Variablen können Sie in VBScript mit den Schlüsselworten `Dim`, `Private` und `Public` deklarieren. Da VBScript nur den Datentyp `Variant` kennt, können Sie bei der Deklaration keinen Datentyp angeben. Die Art der Deklaration und damit die Lebensdauer und Gültigkeit der Variable hängt davon ab, wo Sie diese deklarieren. Die Syntax der Deklaration ist:

```
{Dim | Private | Public} Name1 [, Name2] [...]
```

Lokale Variablen

Innerhalb von Funktionen oder Prozeduren können Sie Variablen nur mit `Dim` deklarieren. Eine solche Variable gilt nur in der Funktion bzw. Prozedur und lebt nur, so lange die Funktion bzw. Prozedur läuft:

```
<%
Sub Test()
   Dim i ' Diese Variable gilt nur innerhalb der Prozedur
   i = 10
   Response.Write i
```

```
End Sub
%>
```

Wenn Sie versuchen, eine in einer Prozedur deklarierte Variable außerhalb der Prozedur zu verwenden, erhalten Sie (bei eingeschaltetem Zwang zur expliziten Variablendeklaration) einen Laufzeitfehler »Variable ist nicht definiert«:

```
<%
Sub Test1()
   Dim i ' Diese Variable gilt nur innerhalb der Prozedur
   i = 10
End Sub

Sub Test2()
   Response.Write i ' erzeugt einen Laufzeitfehler
   ' »Variable ist nicht definiert«
   ' wenn die Prozedur aufgerufen wird
End Sub

%>
```


Modulglobale Variablen

Sie können Variablen auch auf einer höheren Ebene deklarieren, nämlich einfach innerhalb eines Skripts, aber außerhalb von Prozeduren oder Funktionen. Solche Variablen werden im Allgemeinen als modulglobale Variablen bezeichnet, weil sie im gesamten »Modul« (in unserem Fall ist das die ASP-Datei) gelten.

Für diese Deklaration können Sie die Schlüsselwörter `Dim`, `Private` und `Public` verwenden, es macht aber keinen Unterschied welches Sie verwenden ☺. Außerhalb von Funktionen/Prozeduren in Skripten deklarierte Variablen gelten immer in der gesamten ASP-Datei, aber nur in dieser. Auch wenn Sie eine Variable mit `Public` deklarieren, gilt diese nicht in anderen ASP-Dateien. Für solche programmglobalen Variablen müssen Sie eine andere Technik verwenden (wie ich es im nächsten Abschnitt beschreibe). `Public` ist in VBScript eigentlich für Klassen vorgesehen (siehe Seite 181) und wird in einfachen Skripten lediglich unterstützt. Variablen, die in allen Skripten, Prozeduren und

Funktionen einer ASP-Datei gelten sollen, sollten eigentlich mit `Private` deklariert werden, um deren Gültigkeitsbereich klar anzugeben. Die Deklaration mit `Dim` ist aber identisch, weswegen Sie in vielen ASP-Dateien auch `Dim` sehen.

Das folgende Beispiel deklariert eine private Variable, die in einem Skript initialisiert, in einer Prozedur gesetzt und in einer anderen Prozedur abgefragt wird:

```
<%
Private i
i = 1

Sub Test1()
   i = 10
End Sub

Sub Test2()
   Response.Write i
End Sub

Test1
Test2
%>
```

Die Lebensdauer einer modulglobalen Variablen reicht übrigens – anders als in normalen Programmiersprachen – nur so lange, wie die ASP-Datei abgearbeitet wird. Modulglobale Variablen werden vom IIS zerstört, wenn die Abarbeitung der ASP-Datei beendet ist, und neu angelegt, wenn eine ASP-Datei neu eingelesen wird. Sie können in solchen Variablen also keine Werte speichern, die beim nächsten Aufruf der ASP-Seite wiederverwendet werden sollen. Dazu müssen Sie in ASP immer programmglobale Variablen verwenden.

Programmglobale Variablen

Programmglobale Variablen, die ihren Wert über die »gesamte« Lebensdauer eines Programms speichern, können Sie in ASP nicht wie normale Variablen deklarieren. Solche Werte müssen Sie entweder im `Session`- oder im `Application`-Objekt speichern. Diese Objekte, die

Ihnen ASP zur Verfügung stellt (und die in Kapitel 8 noch näher erläutert werden), besitzen u. a. je eine Auflistung, in die Sie beliebige Werte speichern können. Die Syntax dazu ist:

```
Session("Schlüssel") = Wert
Application("Schlüssel") = Wert
```

Schlüssel ist ein von Ihnen definierter String, über den Sie später auf die gespeicherten Werte zugreifen können. Das folgende Listing zeigt dies am Beispiel eines Zugriffszählers für eine ASP-Datei:

```
<%
 Application("HitCounter") = Application("HitCounter") + 1
 Response.Write "Auf diese ASP-Datei wurde " & _
    "insgesamt " & Application("HitCounter") & _
    " mal zugegriffen."
%>
```

Der Unterschied zwischen der Speicherung im `Application`- und im `Session`-Objekt ist, dass Werte, die im `Application`-Objekt gespeichert werden, in allen ASP-Seiten *aller* Sitzungen gültig sind. Ändert eine ASP-Seite in der Sitzung eines Benutzers eine `Application`-Variable, kann dieser Wert in einer ASP-Seite in einer Sitzung eines anderen Benutzers abgerufen werden.

`Session`-Variablen gelten dagegen nur in der Sitzung, in der sie erzeugt wurden. Eine Sitzung wird immer dann automatisch erzeugt, wenn ein Browser eine beliebige Datei eines Webordners abruft, für den die Option »Sitzungsstatus aktivieren« eingeschaltet ist. Eine Sitzung bleibt für diesen einen Browser so lange geöffnet, wie der Benutzer Dateien abruft, und noch 20 Minuten nachdem der Benutzer die letzte Datei abgerufen hat. Sitzungen werden noch näher im nächsten Kapitel erläutert.

7.5.4 Namenskonventionen für Variablen und Objekte

In Visual Basic und in VBScript setzen Programmierer üblicherweise eine Namenskonvention zur Benennung von Variablen und Objekten ein. Diese von Greg Reddik entwickelte Konvention legt u. a. fest, dass Bezeichner für Variablen und Objekte immer mit einem dreistelligen, kleingeschriebenen Typkürzel beginnen. Der eigentliche Bezeichner

wird dann mit einem Großbuchstaben begonnen. Eine Variable, die eine Positionsangabe in Form eines Longinteger-Werts speichern soll, wird zum Beispiel so benannt: *lngPosition*. So können Sie schon am Namen einer Variablen erkennen, dass es sich um eine solche handelt. Dass Sie damit auch gleich den Datentyp der Variablen auslesen können, erweist sich in der Praxis häufig als enormer Vorteil.

Für C#, neben VB.NET eine der wichtigsten Programmiersprachen von .NET, werden solche Typkennzeichen im Allgemeinen nicht verwendet. Ich habe dieses Thema einige Male in den C#-Newsgroups diskutiert mit dem Ergebnis, dass eigentlich kein Programmierer mehr Typkennzeichen für diese Sprache anwenden will. Die Gründe dafür sind vielfältig und hängen u. a. damit zusammen, dass C# eine typsichere Sprache ist und deshalb keine Typkennzeichen benötigt (einer Stringvariablen können Sie z. B. keine Zahl zuweisen, was in VBScript ohne weiteres geht) und dass die enorm vielen Datentypen (u. a. die Klassen) des .NET-Framework zu einer unübersichtlichen Anzahl von Typkennzeichen führen würde. Ich habe dann darüber nachgedacht, diese Typkennzeichen auch nicht mehr in VBScript und Visual Basic anzuwenden. Weil die Reddik-Konvention bei diesen typunsicheren Sprachen aber sehr hilfreich ist und von fast allen Programmierern angewendet wird, habe ich mich entschlossen, die Konvention für dieses Buch einzuhalten.

Tabelle 7.3 listet die gängigen Typkürzel für einfache Variablen auf.

Datentyp	Typkürzel	Beispiel
`Boolean`	bln	blnOK
`Byte`	byt	bytAge
`Currency`	cur	curFee
`Date`	dtm (**D**ate**T**i**m**e)	dtmBirthday
`Double`	dbl	dblDiscount
`Integer`	int	intYear
`Long`	lng	lngIndex

Datentyp	Typkürzel	Beispiel
Object	obj	objWord
Single	sng	sngPrice
String	str	strFilename
Variant	var	varTemp

Tabelle 7.3: Die in VBScript gängigen Typkennzeichen für Variablen

Wenn Sie ein Array deklarieren, sollten Sie dem Typkennzeichen noch ein »a« voranstellen:

```
Dim aintZahlen()
```

Die Reddik-Konvention definiert neben den Typkennzeichen für Variablen noch eine Menge Typkennzeichen für Objekte, wie zum Beispiel Steuerelemente. Gerade in ASP finde ich es sehr wichtig, dass man einem Steuerelement-Bezeichner ansieht, dass es sich um ein Steuerelement handelt. Aber auch in »echten« Programmiersprachen wie zum Beispiel C# werde ich nicht auf die Typkennzeichen für Steuerelemente verzichten. Nur über diese kann ich ein Steuerelement von einer normalen Variablen unterscheiden. Ein anderer Vorteil ist, dass ich über Typkennzeichen unterschiedliche Steuerelemente, die in einem Zusammenhang stehen, prinzipiell gleich benennen kann, aber eben über das Typkennzeichen unterscheide.

Tabelle 7.4 listet deswegen die Typkennzeichen der Steuerelemente auf, die in ASP bzw. HTML von mir und vielen anderen Programmierern verwendet werden.

Steuerelement	Typkürzel
Textbox, Textarea	txt
ComboBox (einzeilige Liste)	cbo
Listbox (mehrzeilige Liste)	lst
CheckBox	chk
OptionButton	opt
Button, Submit, Reset	btn

Tabelle 7.4: Die Typkürzel für HTML-Steuerelemente

7.5.5 Arrays

Der in VBScript verwendete Begriff *Array* steht für ein Feld aus mehreren Variablen. In C/C++ würde man ein solches Feld als *Vektor* bezeichnen. Arrays erlauben das zusammenhängende Speichern mehrerer gleichartiger Informationen. Über einen Index können Sie auf die Einzelinformationen zugreifen. VBScript unterscheidet statische und dynamische Arrays. Dynamische Arrays werden hier nicht beschrieben, weil die in Kapitel 8 beschriebene `Dictionary`-Klasse wesentlich flexibler und beim Suchen sogar schneller ist, als ein statisches oder dynamisches Array.

Statische Arrays werden folgendermaßen deklariert:

```
{Dim | Private | Public} Name (Obere Grenze Dimension 1 _
   [, Obere Grenze Dimension 2] [...])
```

Der Gültigkeitsbereich entspricht dem von Variablendeklarationen.

Obere Grenze Dimension n gibt den oberen Grenzindex der Datenfelder für jede Dimension an. Wenn Sie z. B. ein eindimensionales Array erzeugen wollen, dessen Index von 0 bis 3 reicht, verwenden Sie eine dem folgenden Beispiel ähnliche Syntax:

```
Dim aintZahlen(3)
```


Beachten Sie, dass Sie nicht die Anzahl der Elemente angeben, sondern den oberen Grenzindex. Der Index eines Arrays, das wie im obigen Beispiel deklariert ist, reicht also von 0 bis 3. Das Array speichert demnach vier Elemente. Anders als in Visual Basic 6 können Sie in VBScript aber auch ein Array mit dem oberen Grenzindex 0 deklarieren (das dann eben nur ein Element speichert). Dadurch werden einige Probleme vermieden, die mit dieser Art der Deklaration in Visual Basic 6 besonders bei dynamischen Arrays auftreten.

Der Zugriff auf ein Array erfolgt über die Angabe des Index in Klammern. Der folgende Quellcode schreibt Zahlen in das Array und gibt den Inhalt danach aus:

```
aintZahlen(0) = 10
aintZahlen(1) = 20
```

```
aintZahlen(2) = 30
aintZahlen(3) = 40

Dim i
For i = 0 to 3
   Response.Write aintZahlen(i) & "<br>"
Next
```

Für mehrdimensionale Arrays geben Sie den oberen Grenzindex der weiteren Dimensionen durch Kommata in der Klammer getrennt an. Ein zweidimensionales Array erzeugen Sie z. B. so:

```
Dim aintZahlen(2, 1)
```

Der Zugriff erfolgt analog:

```
aintZahlen(0, 0) = 11
aintZahlen(1, 0) = 12
aintZahlen(2, 0) = 13
aintZahlen(0, 1) = 21
aintZahlen(1, 1) = 22
aintZahlen(2, 1) = 23
```


In VBScript sind maximal 60-dimensionale Arrays möglich. Mehrdimensionale Arrays werden allerdings heutzutage kaum noch benötigt. Stattdessen verwenden viele Programmierer lieber `Collection`- oder `Dictionary`-Objekte.

Über die Funktionen `LBound` und `UBound` können Sie die Grenz-Indizes des Arrays ermitteln. Beide Funktionen sind wichtig bei der Übergabe von Arrays an Prozeduren und bei der Auswertung von dynamisch erstellten Arrays.

Initialisieren von Arrays

Leider können Sie statische Arrays in VBScript nicht initialisieren. Also müssen Sie die voreingestellten Werte eines statischen Arrays im Programmcode setzen. Alternativ können Sie statt eines statischen Arrays einfach ein dynamisches Array mit Hilfe der `Array`-Funktion erstellen. Da diese einen einfachen `Variant`-Datentyp (und kein Array) zurückgibt, müssen Sie leider eine entsprechend deklarierte Variable verwenden:

```
Dim astrFarben
astrFarben = Array("Blau", "Rot", "Grün")
```

Strukturen

Strukturen werden von VBScript nicht unterstützt. Verwenden Sie stattdessen Klassen (Seite 181).

7.5.6 Symbolische Konstanten

Symbolische Konstanten deklarieren Sie mit dem Schlüsselwort `Const`:

```
[Private] Const Name1 = Ausdruck
   [, Name2 = Ausdruck] [...]
```

Der Gültigkeitsbereich einer symbolischen Konstante entspricht dem einer Variablen. Innerhalb von Funktionen oder Prozeduren deklarieren Sie die Konstante einfach nur mit `Const`. Wenn Sie eine Konstante auf Modulebene deklarieren, besitzt diese per Voreinstellung den Gültigkeitsbereich `Private`. Auf das Schlüsselwort `Private` können Sie bei der Deklaration also verzichten. Das folgende Beispiel deklariert eine private symbolische Konstante für den Wert PI und eine lokale Konstante für einen linken Rand:

```
Private Const PI = 3.1415927

Sub Test()
 ' lokale Konstante LEFT_MARGIN
 Const LEFT_MARGIN = 100
 ...
End Sub
```

Der Bezeichner einer Konstanten wird üblicherweise in Großbuchstaben geschrieben.

Vordefinierte VBScript-Konstanten

VBScript besitzt bereits sehr viele symbolische Konstanten. Der Name dieser Konstanten beginnt immer mit »vb«. Die Konstante `vbRed` steht z. B. für den Zahlwert der Farbe Rot. In der VBScript-Referenz sind diese Konstanten beschrieben.

Tipp

Setzen Sie grundsätzlich konsequent symbolische Konstanten ein, wenn Sie nur die Ahnung haben, ein eingesetzter Wert könnte sich irgendwann einmal ändern. So können Sie eine später erforderliche Änderung des Wertes sehr leicht über eine Änderung der symbolischen Konstanten erledigen. Außerdem können Sie symbolische Konstanten später auch ohne große Probleme in Variablen umwandeln, wenn Sie z. B. die gespeicherten Werte für den Anwender editierbar machen wollen. Deklarieren Sie symbolische Konstanten, die für einige oder alle ASP-Dateien gelten, in einer Include-Datei, die Sie in die jeweiligen ASP-Dateien einbinden (Seite 196).

7.5.7 Verzweigungen und Schleifen

VBScript kennt natürlich auch die gängigen Verzweigungen und Schleifen.

If...Then

Die If-Then-Verzweigung ist in VBScript gegenüber anderen Programmiersprachen um optionale `ElseIf`-Blöcke erweitert:

```
If Bedingung1 Then
   [Anweisungsblock 1]]
[ElseIf Bedingung2 > Then
   [Anweisungsblock 2]]
[...]
[Else
   [Anweisungsblock n]]
End If
```

Der optionale `ElseIf`-Block kann beliebig oft in die `If-Then`-Verzweigung eingebaut werden. Die Bedingung eines `ElseIf`-Blocks wird nur dann geprüft, wenn keine der Bedingungen der vorstehenden Blöcke erfüllt war. Dasselbe gilt für den `Else`-Block, nur dass hier keine Bedingung mehr überprüft wird. Eine einfache If-Then-Verzweigung, die überprüft, ob heute Wochenende oder ein Arbeitstag ist, sieht so aus:

```
If (Weekday(Now) = 7) Or (Weekday(Now) = 1) Then
   ' Samstag oder Sonntag
   Response.Write "Es ist Wochenende, " & _
```

```
      "nur Buchautoren müssen arbeiten."
   ' Anmerkung der Korrektorin:
   ' Stimmt üüüüberhaupt gaaaar nicht! ☺
Else
   Response.Write "Es ist kein Wochenende, " & _
      "alle müssen arbeiten."
End If
```

Kurzform der If-Then-Verzweigung in einer Zeile

Sie können die `If-Then`-Verzweigung auch in eine Zeile schreiben, dann allerdings ohne `ElseIf`-Blöcke:

```
If Bedingung Then Anweisung [Else Anweisung]
```

Mit einem Trick können Sie auch mehrere Anweisungen in einem Block dieser Verzweigung verwenden: Trennen Sie die einzelnen Anweisungen einfach durch einen Doppelpunkt.

Select Case

Die `Select-Case`-Verzweigung überprüft einen Ausdruck auf mehrere mögliche Ergebnisse:

```
Select Case Testausdruck
   [Case Ausdrucksliste 1
      [Anweisungsblock 1]]
   [Case Ausdrucksliste 2
      [Anweisungsblock 2]]
   [...]
   [Case Else
      [Anweisungsblock n]]
End Select
```

Auch die `Select-Case`-Verzweigung ist gegenüber anderen Programmiersprachen erheblich erweitert. So können Sie im Testausdruck einen beliebigen arithmetischen oder logischen Ausdruck angeben (andere Programmiersprachen erlauben hier nur Ganzzahl-Datentypen).

Die einzelnen `Case`-Blöcke überprüfen, ob der Ergebniswert des Testausdrucks mit einem der Werte in der Ausdrucksliste übereinstimmt. Die Ausdrucksliste kann, durch Kommata getrennt, einzelne Werte

enthalten. Bereichsangaben (*Wert1* `To` *Wert2)*, die in Visual Basic möglich sind, sind in VBScript nicht erlaubt. Im Gegensatz zu manchen anderen Programmiersprachen können Sie in VBScript komplexe Ausdrücke und auch Strings in die Ausdrucksliste einarbeiten. In der Praxis werden jedoch normalerweise lediglich einfache Werte und Listen von Werten eingesetzt.

Ist einer der Werte in der Ausdrucksliste mit dem Wert des Testausdrucks identisch, wird der zugehörige Anweisungsblock ausgeführt. Danach ist die `Select-Case`-Verzweigung, anders als z. B. in C, beendet. Der optionale `Case-Else`-Block wird ausgeführt, wenn keiner der vorherigen Blöcke ausgeführt wurde:

```
Select Case intZahl
  Case 1, 2, 3
    Response.Write "Die Zahl ist gleich 1, 2 oder 3"
  Case 4
    Response.Write "Die Zahl ist gleich 4"
  Case 5, 6
    Response.Write "Die Zahl ist gleich 5 oder 6"
  Case Else
    Response.Write "Die Zahl ist nicht 1, 2, 3, " & _
       "4, 5 oder 6"
End Select
```


Die Do-Schleife

Mit der `Do`-Schleife können Sie kopfgesteuerte, fußgesteuerte und Endlosschleifen erzeugen:

```
Do [{While | Until} Bedingung]
   Anweisungen
Loop [{While | Until} Bedingung]
```

Eine Schleife, die eine Bedingung im Kopf überprüft, zeigt das folgende Listing:
Beispiel: Schleifen, solange der Inhalt einer Variablen kleiner 4 ist:

```
Dim intCount
intCount = 1
Do While intCount < 4
   Response.Write intCount
```

```
   intCount = intCount + 1
Loop
```

Fußgesteuerte Schleifen werden, wie Sie ja wissen, mindestens einmal durchlaufen, auch wenn die Bedingung beim Eintritt in die Schleife falsch ist.

```
Dim intCount
intCount = 10
Do
   Response.Write intCount
   intCount = intCount + 1
Loop Until intCount > 3
```

»Endlosschleifen« sind mit der `Do`-Schleife auch möglich. Für diese Schleifen lassen Sie die Bedingung einfach weg. Damit die Schleife nicht wirklich endlos läuft, wird üblicherweise innerhalb der Schleife eine Bedingungen überprüft und die Schleife mit `Exit Do` abgebrochen.

```
Dim intCount
intCount = 1
Do
   Response.Write intCount
   If intCount = 3 Then
      Exit Do
   End If
   intCount = intCount + 1
Loop
```

Diese »Endlosschleife« können Sie verwenden, wenn die Bedingung zu komplex ist, um im Kopf oder Fuß der Schleife geprüft zu werden, oder wenn beim Eintreten der Bedingung zusätzliche Anweisungen ausgeführt werden sollen. Prüfen Sie die Bedingung einfach innerhalb der Schleife und verlassen Sie die Schleife dann mit `Exit Do`.

Endlosschleifen und der IIS

Wenn Sie in einer ASP-Seite einmal eine Endlosschleife produzieren (eine Schleife, deren Abbruchbedingung niemals wahr wird), erreichen Sie damit, dass sich der IIS zunächst »aufhängt«. Bei der endlosen Abarbeitung der Schleife benötigt der Prozess, der die Ausfüh-

rung der ASP.DLL übernimmt (INETINFO.EXE bzw. DLLHOST.EXE, je nach Isolationsebene des Webordners), etwa 90-100% Prozessorzeit. Die Ausführung des Skripts wird allerdings automatisch vom IIS abgebrochen, wenn das Skript-Timeout abgelaufen ist. Das Skript-Timeout, das normalerweise auf 90 Sekunden eingestellt ist, verwalten Sie über die Eigenschaften des Webordners. Warten Sie im Falle einer Endlosschleife (die bei Ihnen wahrscheinlich genauso häufig vorkommt wie bei mir) also einfach, bis das Timeout abgelaufen ist. Die aktuelle CPU-Nutzung der DLLHOST.EXE bzw. der INETINFO.EXE können Sie über den Windows Task-Manager ermitteln.

Die While-Schleife

VBScript kennt noch eine `While`-Schleife, die jedoch nahezu identisch ist mit der `Do`-Schleife:

```
While Bedingung
   Anweisungen
Wend
```

Im Gegensatz zur `Do`-Schleife kann die `While`-Schleife nicht mittendrin abgebrochen werden.

Die For-Next-Schleife

Die `For-Next`-Schleife ist die Zählschleife von VBScript. Diese Schleife zählt eine nummerische Variable automatisch hoch, bis diese einen Wert besitzt, der größer ist als der in der Schleife angegebene Endwert:

```
For Zähler = Startwert To Endwert [Step Schrittweite]
   Anweisungen
Next
```

Die Schrittweite ist standardmäßig 1. Mit dem optionalen `Step` können Sie die Schrittweite aber auch auf größere oder negative Werte oder auch auf Dezimalwerte setzen. So können Sie z. B. von 3 nach 1 rückwärtszählen:

```
Dim lngCounter
For lngCounter = 3 To 1 Step -1
   Response.Write lngCounter
Next
```

Die For-Next-Schleife können Sie mit Exit For explizit verlassen.

Die For-Each-Schleife

Mit der For-Each-Schleife können Sie alle Elemente eines Arrays oder einer Auflistung (Collection oder Dictionary) ohne Kenntnis der Anzahl der gespeicherten Elemente durchgehen.

```
For Each Element In {Array | Collection}
   [Anweisungen]
Next [Element]
```

Element ist eine Variable, der innerhalb der Schleife automatisch das jeweilige Element der Liste zugewiesen wird.

Das folgende Beispiel geht ein Array mit For Each durch:

```
Dim i, varElement
Dim aintZahlen(9)
' Array füllen
For i = 0 To 9
  aintZahlen(i) = i
Next
' Array mit For Each durchgehen
For Each varElement In aintZahlen
   Response.Write varElement
Next
```

Eine For-Each-Schleife können Sie, wie eine For-Next-Schleife, mit Exit For vor dem impliziten Beenden explizit beenden.

7.5.8 Die With-Anweisung

Über die With-Anweisung können Sie ab VBScript 5.0 (IIS 5) die Elemente eines Objekts in Anweisungen verwenden, ohne das Objekt jedes Mal neu angeben zu müssen.

```
With Objekt
   [Anweisungen]
End With
```

Methoden und Eigenschaften des Objekts geben Sie innerhalb des With-Blocks nur mit einem Punkt als Präfix an. Damit erleichtern Sie sich die Schreibarbeit, aber dem Compiler auch einiges an Aufwand.

Das folgende Beispiel verwendet einige Eigenschaften und Methoden des `Response`-Objekts:

```
With Response
   .Expires = 0
   .Buffer = True
   For i = 1 To 1000
      .Write i & "<br>"
   Next
  .Flush
End With
```

Sie können die `With`-Anweisung auch verschachteln. Das folgende Beispiel erzeugt einen Cookie[1] auf dem Rechner des Benutzers über die `Cookies`-Auflistung des `Response`-Objekts und setzt dann im inneren `With`-Block Eigenschaften des Cookie:

```
With Response
   .Cookies("LastAccess") = Now
   With .Cookies("LastAccess")
      .Expires = Now + 100
      .Path = "\"
      .Domain = "Nitty-Gritty.de"
   End With
End With
```


7.5.9 Prozeduren und Funktionen

VBScript kennt neben Funktionen (die einen Wert, ein Array oder ein Objekt zurückgeben) auch Prozeduren. Eine Prozedur ist quasi eine Funktion, die nichts zurückgibt.

Prozeduren

Prozeduren deklarieren Sie in VBScript innerhalb der ASP-Datei (oder in einer externen Include-Datei, wie ich es auf Seite 196 beschreibe), idealerweise im Head-Bereich, über die folgende Syntax:

1. Ein Cookie ist ein Wert, der auf dem Rechner des Internetbenutzers gespeichert und in späteren Sitzungen wieder ausgelesen werden kann.

```
Sub Name([Argumentliste])
   Anweisungen
End Sub
```

In der optionalen Argumentliste können Sie Argumente deklarieren, die dann beim Aufruf der Prozedur an diese übergeben werden müssen. Mehrere Argumente trennen Sie durch Kommata.

Eine einfache Prozedur ohne Argumente sieht (in einer ASP-Datei inkl. Aufruf der Prozedur) z. B. so aus:

```
<%@Language="VBScript"%>
<%Option Explicit%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<title>Prozeduren und Funktionen</title>

<%
 ' Deklaration der Prozeduren und Funktionen
 Sub SayHello()
    Response.Write "Hello World"
 End Sub
%>

</head>
<body>

<%
 ' Aufruf der einfachen Prozedur
 SayHello
%>

</body>
</html>
```

Eine Prozedur, die einen Text mit einem Zeilenumbruch oder in einem Absatz ausgibt, zeigt das folgende Listing:

```
Sub WriteLine(Text, Break)
   If Break = 1 Then
      Response.Write "<p>" & Text & "</p>"
   Else
      Response.Write Text & "<br>"
   End If
End Sub
```

Den Aufruf dieser Prozedur zeigt die folgende Quellcode-Zeile:

```
WriteLine "Ich bin ein Absatz", 1
```

Funktionen

Eine Funktion wird folgendermaßen deklariert:

```
Function Name([Argumentliste])
 Anweisungen
 Name = Wert
End Function
```

Wenn Sie eine Funktion entwickeln, schreiben Sie das, was Sie zurückgeben wollen, in eine implizit erstellte Variable, die denselben Namen trägt wie die Funktion. An welcher Stelle Sie die Rückgabe in diese Variable schreiben, ist vollkommen unerheblich. Schreiben Sie keinen Wert, gibt die Funktion einen Leerwert zurück (bei Zahlen ist das die 0, bei Strings ein Leerstring).

Die folgende Funktion gibt eine auf eine bestimmte Anzahl Dezimalstellen aufgerundete Zahl zurück:

```
Function RoundUp(Value, Decimals)
 Value = Value * 10 ^ Decimals
 If Value - Int(Value) > 0 Then
    RoundUp = (Int(Value) / 10 ^ Decimals) + _
       (1 / 10 ^ Decimals)
 Else
    RoundUp = Int(Value) / 10 ^ Decimals
 End If
End Function
```

Um das Beispiel vollständig zu machen, folgt nun noch ein Beispiel für den Aufruf dieser Funktion (aber das kennen Sie ja bereits):

```
Response.Write "<p>" & RoundUp(1.234, 2) & "</p>"
```

Call by Value und Call by Reference

Wie die meisten anderen Programmiersprachen auch kennt VBScript die Übergabearten »By Reference« und »By Value«. Werden Argumente »By Reference« deklariert und Sie übergeben eine Variable an dieses Argument, kann die Prozedur bzw. Funktion die Variable nach außen hin verändern. Leider ist die Voreinstellung der Argumentübergabe »By Reference«. Sie können die Übergabeart über die Schlüsselwörter `ByVal` und `ByRef` aber auch explizit einstellen.

Da die Übergabe »By Reference« zu schwer lokalisierbaren logischen Fehlern führen kann, nämlich dann, wenn Sie innerhalb einer Prozedur bzw. Funktion ein Argument modifizieren und damit eine an dieser Stelle übergebene Variable nach außen hin verändern, sollten Sie grundsätzlich »By Value« übergeben, was die Angabe von `ByVal` *vor jedem Argument erforderlich macht. Nur wenn Sie Referenzargumente explizit benötigen, übergeben Sie »By Reference«. In diesem Fall sollten Sie zur besseren Übersicht das Schlüsselwort* `ByRef` *verwenden.*

7.5.10 Klassen

Ab VBScript 5.0 können Sie auch eingeschränkt objektorientiert programmieren und einfache Klassen deklarieren. Eine Klassendeklaration kann private und öffentliche Eigenschaften und Methoden enthalten. Eigenschaften können als einfache Werte oder als so genannte Property-Prozeduren implementiert werden. Mit Property-Prozeduren können Sie Eigenschaften implementieren, die schreib- oder lesegeschützt sind oder die ihren Wert beim Schreiben überprüfen oder beim Lesen berechnen. Eine Vererbung von Klassen, wie in anderen Sprachen, ist nicht möglich.

Ich beschreibe Klassen hier nur so, dass Sie diese anwenden können. Die notwendigen OOP-Grundlagen kann ich natürlich nicht beschreiben. Falls Sie zu wenig OOP-Erfahrung besitzen, um diesen Abschnitt zu verstehen, lesen Sie meinen Artikel »OOP-Grundlagen«, den Sie in den zusätzlichen Kapiteln meines Visual Basic-Buchs auf der Seite www.nitty-gritty.de *finden. Hilfreich ist auch der Artikel »OOP mit Visual Basic 6«, der noch genauer auf die speziellen Property-Prozedur-Eigenschaften eingeht.*

Die Grundform einer Klasse ist folgende:

```
Class Name
 Deklarationen
End Class
```

In VBScript können Sie Klassen zwar in ASP-Token deklarieren. Da dies in VB.NET und C# jedoch nicht möglich ist, sollten Sie Klassen immer in einem `script`*-Tag deklarieren.*

Einfache Eigenschaften deklarieren Sie in der Klasse wie Variablen. Verwenden Sie das Schlüsselwort `Public`, wenn die Eigenschaft von außen zugreifbar sein soll, ansonsten `Private`. Private Elemente einer Klasse können nur innerhalb der Klasse verwendet werden.

Die folgende Klasse soll die Daten von Personen speichern:

```
Class Person
 Public Vorname
 Public Nachname
 Public Geburtsdatum
 Public Ort
End Class
```

Ein Objekt dieser Klasse erzeugen Sie über den `New`-Operator:

```
Dim p1, p2
Set p1 = New Person
p1.Vorname = "Zaphod"
p1.Nachname = "Beeblebrox"
Response.Write p1.Vorname & " " & p1.Nachname
```

Methoden implementieren Sie wie Funktionen und Prozeduren. Wie in der OOP üblich können Methoden auf alle Elemente einer Klasse zugreifen, auch wenn diese privat sind. Das folgende Listing erweitert die Klasse um eine Methode, die das Alter berechnet:

```
Class Person
  Public Vorname
  Public Nachname
  Public Geburtsdatum
  Public Function GetAlter()
     GetAlter = DateDiff(Now, Geburtsdatum, "yyyy")
  End Function
End Class
```

Mit Property-Prozeduren können Sie Eigenschaften erzeugen, die von außen zwar wie ganz normale Eigenschaften aussehen, intern aber über eine spezielle Schreib- und Lesemethode implementiert sind. Innerhalb dieser Methoden können Sie programmieren, weswegen Ihnen alles offen steht. Sie können beim Schreiben z. B. dafür sorgen, dass nur gültige Werte geschrieben werden können. Die normale Syntax dieser Prozeduren ist:

```
' Schreib-Prozedur
[Public [Default] | Private] Property Let Name(Wert)
   Anweisungen
End Property

' Lese-Prozedur
[Public [Default] | Private] Property Get Name()_
   Anweisungen
End Property
```

Die Syntax einer Property-Prozedur ist in Visual Basic.NET etwas anders (und in C# komplett anders). Sie können dieses Beispiel also nicht auf VB.NET übertragen. In den Buchbeispielen zu diesem Kapitel finden Sie den hier dargestellten Quellcode auch in der VB-NET-Variante. Schauen Sie einfach dort nach.

Handelt es sich um eine Objekt-Eigenschaft, müssen Sie in der Schreib-Prozedur statt `Let` das Schlüsselwort `Set` verwenden.

Das folgende Beispiel erweitert die Personenklasse um die Überprüfung des Geburtsdatums auf ein Datum, das nicht in der Zukunft liegt, und auf die Rückgabe des vollen Namens in einer Eigenschaft:

```
Class Person
  Public Vorname
  Public Nachname
  ' Private Variable für die Speicherung
  ' des Geburtsdatums
  Private dtmGeburtsdatum

  ' Schreib-Prozedur der Eigenschaft Geburtsdatum,
  ' die das Geburtsdatum überprüft
  Public Property Let Geburtsdatum(Datum)
   If Datum <= Now Then
      dtmGeburtsdatum = Datum
   Else
      dtmGeburtsdatum = 0
   End If
  End Property

  ' Lese-Prozedur für das Geburtsdatum
  Public Property Get Geburtsdatum()
   If dtmGeburtsdatum > 0 Then
      Geburtsdatum = dtmGeburtsdatum
   Else
      Geburtsdatum = "invalid"
   End If
  End Property

  ' Methode zur Berechnung des Alters
  Public Function GetAlter()
     GetAlter = DateDiff(Now, Geburtsdatum, "yyyy")
  End Function
  ' Schreibgeschützte Eigenschaft zur
  ' Ermittlung des vollen Namens
  Public Property Get Name()
     Name = Vorname & " " & Nachname
  End Property
End Class
```

Wie Sie dem Beispiel entnehmen können, benötigen Sie eine private Eigenschaft, die den Wert der *Geburtsdatum*-Eigenschaft speichert. Im Beispiel überprüft die Schreib-Prozedur den übergebenen Wert und schreibt im Fehlerfall o in diese private Variable. Die Leseprozedur überprüft den Wert und gibt im Fehlerfall »invalid« zurück. Bei der *Name*-Eigenschaft fehlt die Schreibprozedur. Damit ist diese Eigenschaft automatisch schreibgeschützt.

Der Zugriff auf solche speziellen Eigenschaften erfolgt wie der auf normale Eigenschaften:

```
Dim p1
Set p1 = New Person
p1.Vorname = "Zaphod"
p1.Nachname = "Beeblebrox"
p1.Geburtsdatum = "12.12.9999"
Response.Write p1.Name & "; Geburtsdatum: " & _
   p1.Geburtsdatum
```

7.5.11 Die VBScript-Funktionen

Da dieses Buch keine VBScript-Referenz ist, kann ich natürlich nicht alle VBScript-Funktionen beschreiben. Ich zeige deshalb hier nur einige wichtige Funktionen.

Eine komplette Referenz zu den VBScript-Funktionen finden Sie in den zusätzlichen Kapiteln auf der Seite zum Buch bei www.nitty-gritty.de.

Stringfunktionen

Viele Stringfunktionen arbeiten mit einem (meist optionalen) Argument `Compare`, über das Sie festlegen können, wie Strings miteinander verglichen werden. Die Übergabe von `vbTextCompare` bewirkt hier, dass Groß- und Kleinschreibung nicht unterschieden wird. `vbBinaryCompare` führt zum Gegenteil. Wenn Sie zwei Strings miteinander vergleichen wollen und Groß-/Kleinschreibung nicht unterscheiden wollen, müssen Sie `StrComp` verwenden: `StrComp(str1, str2, vbTextCompare)`. Diese Funktion gibt o zurück, wenn beide Strings gleich sind. Ein Vergleich mit = verwendet stattdessen den binären Vergleich.

Wichtige Stringfunktionen sind `LTrim`, `RTrim` und `Trim`. Mit diesen Funktionen können Sie ein- oder beidseitig Leerzeichen aus einem String entfernen. `LTrim(" abc ")` ergibt z. B. "abc ", `RTrim(" abc ")` ergibt " abc" und `Trim(" abc ")` gibt "abc" zurück.

Mit der `Left`- und der `Right`-Funktion können Sie einen linken bzw. rechten Teilstring extrahieren. `Left("abcdef", 3)` ergibt z. B. "def". Die `Mid`-Funktion verwenden Sie, wenn Sie einen Teilstring mitten aus einem String auslesen wollen. `Mid("abcdef", 3, 2)` gibt z. B. "cd" zurück.

Mit `Instr` können Sie ermitteln, an welcher Position ein Teilstring in einem String vorkommt: `Instr(1, ";", strRow, vbTextCompare)` ermittelt z. B. die Position des ersten Semikolons in *strRow*. `InstrRev` arbeitet ähnlich, sucht aber von rechts aus gesehen.

Mit der `Replace`-Funktion können Sie Teilstrings in einer Zeichenkette ersetzen: `Replace(strQuelle, strSuchstring, strErsatzstring, vbTextCompare)`. Wenn Sie nichts weiter angeben, gibt `Replace` einen String zurück, in dem alle Such-Teilstrings durch die Ersatz-Zeichenkette ersetzt sind:

```
Response.Write Replace("1aa2aa3aa", "aa", "bb", _
   vbBinaryCompare) ' ergibt "1bb2bb3bb"
```

`Replace` kann darüber hinaus auch nur eine bestimmte Anzahl Teilstrings ersetzen oder ab einem bestimmten Zeichen ersetzen. Der folgende Quellcode ersetzt z. B. einmal ab dem 5. Zeichen:

```
Response.Write Replace("1aa2aa3aa", "aa", "bb", 5, 1)
' ergibt "bb3aa" (ab dem 5. Zeichen eine Ersetzung)
```

Unverständlich ist mir, dass `Replace` *den String links abschneidet, wenn Sie Start angeben, wie das obige Beispiel zeigt.*

Mit der `UCase`-Funktion können Sie einen String in Großschreibung umwandeln, `LCase` wandelt dagegen in Kleinschreibung um. Die `Asc`-Funktion liefert Ihnen den ANSI-Code eines Zeichens. Umgekehrt können Sie mit `Chr` einen Ansi-Code in ein Zeichen umwandeln.

Eine sehr wichtige Funktion ist `Split`. Mit dieser Funktion können Sie einen String in Teilstrings aufteilen, wenn dieser String Trennzeichen

enthält. `Split` gibt die ermittelten Teilstrings in einem Array zurück. Das folgende Beispiel demonstriert die Arbeitsweise von `Split`:

```
Dim astrResult, strElement
astrResult = Split("123aa456aa789", "aa", -1,
  vbTextCompare)
For Each strElement In astrResult
   Response.Write strElement & "<br>"
Next
```

Das Ergebnis dieses Beispiels ist ein String-Array mit den Strings "123", "456" und "789".

Wenn Sie einen mit `Split` zerlegten String (z. B. nach einer Bearbeitung) wieder zusammenfügen wollen, können Sie die `Join`-Funktion verwenden. Um die Arbeitsweise von `Join` zu demonstrieren, verwendet das folgende Beispiel allerdings ein selbst definiertes String-Array:

```
Dim astrResult(3)
astrResult(0) = "Trillian"
astrResult(1) = "Arthur"
astrResult(2) = "Ford"
astrResult(3) = "Zaphod"
Response.Write Join(astrResult, ";")
' ergibt "Trillian;Arthur;Ford;Zaphod"
```

Eine weitere wichtige Stringfunktion ist die `Len`-Funktion, die die Länge eines Strings ermittelt.

Formatierfunktionen

VBScript enthält mittlerweile einige Funktionen zum Formatieren von Werten: `FormatCurrency`, `FormatDateTime`, `FormatNumber` und `FormatPercent` sind spezialisiert auf Währungswerte, Datumswerte, Zahlen und Prozentwerte. Im ersten Argument geben Sie immer den zu formatierenden Wert an. Bei `FormatCurrency`, `FormatPercent` und `FormatNumber` steuert das zweite Argument die Anzahl der Dezimalstellen. Das dritte Argument dieser Funktionen definiert, ob führende Nullen ausgegeben werden sollen, das vierte, ob negative Zahlen in Klammern ausgegeben werden, und das fünfte, ob Tausender-Trennzeichen verwendet werden:

```
Response.Write FormatCurrency(-0.235, 2, False, False)
' ergibt "-,24 DM"
Response.Write FormatCurrency(-0.235, 2, True, False)
' ergibt "-0,24 DM"
Response.Write FormatCurrency(-0.235, 2, True, True)
' ergibt "(0,24 DM)"

Response.Write FormatPercent(0.00123, 2, False, False)
' ergibt ",12%"
Response.Write FormatPercent(-0.00123, 2, True, False)
' ergibt "-0,12%"
Response.Write FormatPercent(-0.00123, 2, False, True)
' ergibt "(,12%)"
Response.Write FormatPercent(-0.00123, 2, True, True)
' ergibt "(0,12%)"
```

Wenn Sie Datumswerte formatieren, übergeben Sie `FormatDateTime` im optionalen zweiten Argument ein benanntes Format:

```
Response.Write FormatDateTime(Now)
' ergibt "29.06.00 12:17:12"
Response.Write FormatDateTime(Now, vbLongDate)
' ergibt "Donnerstag, 29. Juni 2000"
Response.Write FormatDateTime(Now, vbLongTime)
' ergibt "12:17:41"
Response.Write FormatDateTime(Now, vbShortDate)
' ergibt "29.06.00"
Response.Write FormatDateTime(Now, vbShortTime)
' ergibt "12:18"
```


Konvertierungsfunktionen

Obwohl VBScript bei Operationen zumeist automatisch und korrekt konvertiert, ist das explizite Konvertieren manchmal noch notwendig. Berechnungen oder Vergleiche mit Zahlwerten werden häufig nicht korrekt ausgeführt, wenn die Texte nicht explizit konvertiert werden. Wenn Sie die Interpretation des Datentyps eines Varianten nicht der Script-Engine überlassen wollen, verwenden Sie die VBScript-Konvertierfunktionen. Wichtige Funktionen hierbei sind `CCur`, `CDbl`, `CInt`, `CLng`, `CSng`, `CStr`, `CVar` und `CBool`, die einen beliebigen Aus-

druck in den entsprechenden Datentyp umwandeln. Falls der Ausdruck nicht umgewandelt werden kann, resultiert ein Laufzeitfehler. Beachten müssen Sie, dass `CInt` und `CLng` Dezimalzahlen mathematisch korrekt runden. Dieses für uns etwas ungewöhnliche Runden bewirkt, dass z. B. 0,5 auf 0 und 1,5 auf 2 gerundet wird. Wie zu erwarten, wird 0,51 allerdings auf 1 aufgerundet.

Informationsfunktionen

Informationsfunktionen liefern Informationen zu übergebenen Werten oder Variablen. Wichtige Funktionen sind hier `IsNumeric` und `IsDate`, die überprüfen, ob eine übergebene Variable nummerische Daten bzw. ein gültiges Datum enthält.

7.6 Fehlerbehandlung

Laufzeitfehler treten in ASP-Programmen genauso häufig auf wie in normalen Programmen. Besonders bei der Arbeit mit Datenbanken und Dateien sind die verschiedensten Laufzeitfehler möglich. Wird versucht, eine Verbindung zu einer SQL Server-Datenbank aufzubauen, und der SQL Server ist gerade heruntergefahren, meldet ASP beim Verbindungsaufbau einen Fehler:

```
' Öffnen der Verbindung zu Datenbank 'Pubs'
' auf dem lokalen SQL Server
Dim con
Set con = Server.CreateObject("ADODB.Connection")
con.Open "Provider=SQLOLEDB;Data Source=(local);" & _
   "Initial Catalog=Pubs", "sa", ""
```

Wenn Sie dieses Beispiel ausprobieren, sorgen Sie dafür, dass ein eventuell auf Ihrem System installierter SQL Server nicht läuft, weil ansonsten wahrscheinlich kein Fehler angezeigt wird. Haben Sie auch etwas Geduld, ADO braucht einige Zeit um zu erkennen, dass der Server nicht läuft.

Wenn Sie nichts weiter machen, zeigt der IIS unter ASP den Fehler über eine vordefinierte ASP-Seite an. Unter ASP.NET ist nicht der IIS, sondern das .NET-Framework für die Anzeige der Fehlermeldung zuständig (die allerdings von der Konfiguration des Webordners abhängt).

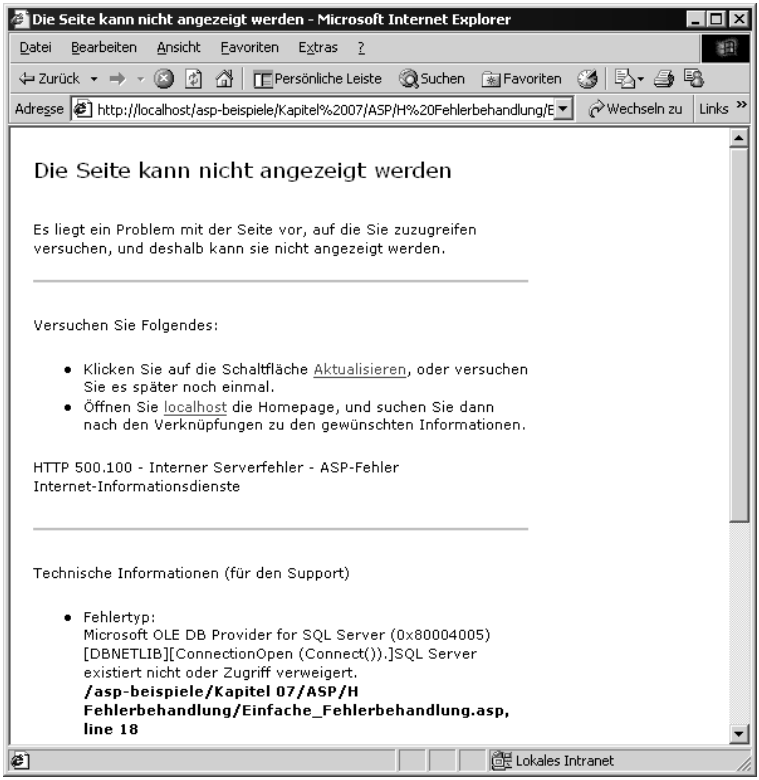


Bild 7.7: Laufzeitfehlermeldung unter ASP

Lassen Sie sich nicht von der etwas unübersichtlichen Fehlermeldung verwirren. Einige Microsoft-Programmierer scheinen Spaß an komplizierten Aussagen zu haben.

Diese interne Fehlerbehandlung können Sie durch eine eigene ersetzen, indem Sie für ASP eine ASP-Seite erzeugen und diese in der Konfiguration der Anwendung mit dem HTTP-Fehler 500 verknüpfen. Standardmäßig ist die Datei 500-100.ASP, die Sie im Ordner \WINNT\HELP\IISHELP\COMMON finden, mit diesem Fehler verknüpft. Diese Datei können Sie dann auch als Basis für Ihre eigenen Default-Fehlermeldungen verwenden.

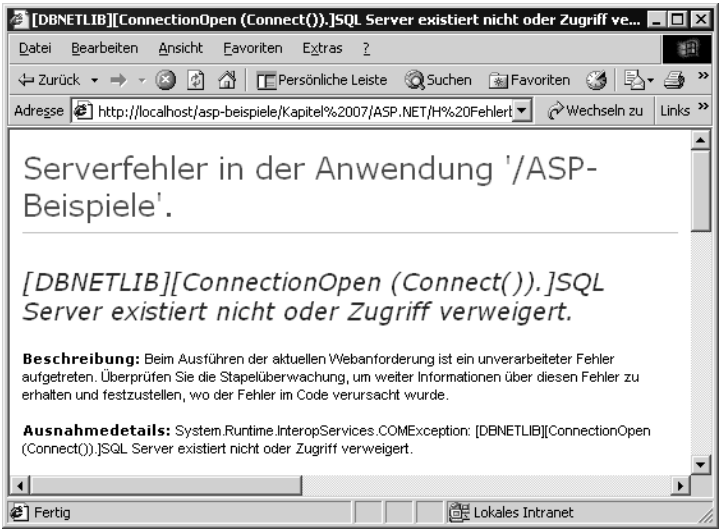


Bild 7.8: Laufzeitfehlermeldung in ASP.NET

ASP.NET lässt Ihnen für Default-Fehlermeldungen mehr Spielraum. Hier können Sie für verschiedene Fehlerklassen eigene Fehlerbehandlungs-Seiten definieren und mit den Fehlerklassen verknüpfen. Bei der Beschreibung der Konfigurationsdatei eines ASP.NET-Webordners in Kapitel 11 zeige ich, wie das geht.

Häufig reicht diese »globale« Fehlerbehandlung schon aus. Manchmal müssen Sie jedoch auf bestimmte Fehler gezielt reagieren. Die dazu notwendige Programmierung zeige ich hier. Da sich die Fehlerbehandlung in ASP und ASP.NET unterscheidet, beschreibe ich beide Varianten separat.

ASP

In ASP müssen Sie die interne Fehlerbehandlung zunächst ausschalten, um eine eigene zu programmieren:

```
On Error Resume Next
```

Diese Anweisung bedeutet so viel wie »Mach bei einem Fehler einfach bei der nächsten Zeile weiter«. Sie platzieren `On Error Resume`

Next vor die Anweisung, bei der Sie einen Fehler erwarten. Direkt hinter der Anweisung werten Sie einen eventuell aufgetretenen Laufzeitfehler über das Err-Objekt aus. In ASP besitzt jeder Laufzeitfehler eine Nummer. Über die Number-Eigenschaft des Err-Objekts können Sie diese Nummer auslesen. Allein die Nummer o steht dafür, dass kein Fehler aufgetreten ist. Fehlernummern können auch negativ sein. Den Text der Fehlermeldung können Sie dann aus der Description- Eigenschaft auslesen. Das folgende Listing zeigt, wie das prinzipiell programmiert wird:

```
' Öffnen der Verbindung zu Datenbank 'Pubs'
' auf dem lokalen SQL Server
Dim con
Set con = Server.CreateObject("ADODB.Connection")

' Interne Fehlerbehandlung ausschalten
On Error Resume Next
' Versuch, die Verbindung zu öffnen
con.Open "Provider=SQLOLEDB;Data Source=(local);" & _
   "Initial Catalog=Pubs", "sa", ""
' Fehler auswerten
If Err.Number <> 0 Then
   ' Fehler zwischenspeichern, da On Error Goto 0 den
   ' Fehler wieder zurücksetzt
   Dim strError
   strError = Err.Description
   ' Interne Fehlerbehandlung wieder einschalten
   On Error Goto 0 ' WICHTIG !
   ' Fehler ausgeben
   Response.Write "Beim Öffnen der Datenbank " & _
      "ist ein Fehler aufgetreten: " & _
      strError
   Response.Write "<p>Benachrichtigen Sie den " & _
      "<a href='mailto:admin@panzerknacker.ag'> " & _
      "Webadiminstrator</a></p>"
End If
' Interne Fehlerbehandlung wieder einschalten
On Error Goto 0 ' WICHTIG !
```

Bild 7.9: Eine eigene Fehlerbehandlung

Wenn Sie jetzt denken, dass das aber aufwändig ist, stimme ich Ihnen zu. Die komplizierte Fehlerbehandlung in ASP war auch immer einer der größten Kritikpunkte.

Sie sollten beachten, dass Fehlernummern auch negative Werte annehmen können. Fragen Sie also immer mit `Err.Number <> 0` ab. Sehr wichtig ist auch das Wieder-Einschalten der internen Fehlerbehandlung.

Schalten Sie die interne Fehlerbehandlung in der Fehlerauswertung und dahinter grundsätzlich immer wieder ein (über `On Error Goto 0`). Wenn Sie vergessen, die interne Fehlerbehandlung wieder einzuschalten, werden weitere Fehler auf der Seite nicht gemeldet. Besonders unglücklich ist das, wenn der erwartete Fehler gar nicht eintritt, dafür aber im folgenden Quellcode der Seite Fehler auftreten. ASP ignoriert dann sogar Fehler, die durch nicht deklarierte Variablen entstehen. Das führt normalerweise dazu, dass die Anwendung nicht mehr korrekt läuft, aber eben keine Fehler meldet. Glauben Sie mir: Nach der Lösung der dadurch verursachten Probleme suchen Sie u. U. tagelang.

Ein kleines Problem ist, dass das `Err`-Objekt mit der Anweisung `On Error Goto 0` zurückgesetzt wird. Sie müssen also Informationen über den aufgetretenen Fehler zwischenspeichern. Im Beispiel lege ich einfach den Fehlertext in einer Variablen ab, die ich in der Fehlermeldung ausgebe.

Bei »Include-Dateien dynamisch einbinden« ab Seite 199 finden Sie eine Möglichkeit, Fehlermeldungen über Dateien auszugeben, die Sie dynamisch in eine ASP-Seite einbinden.

Ein Hinweis noch: Machen Sie sich keine Gedanken um Fehlertexte oder Fehlernummern. Die originalen Fehlertexte sind oft verwirrend, das können Sie nicht ändern. Wenn Sie den Fehlertext aber weglassen, haben Sie selbst keine Information darüber, welcher Fehler aufgetreten ist. Da viele Fehler erst dann auftreten, wenn die Anwendung bereits im Einsatz ist, würde das Weglassen des Fehlertextes das Debuggen erheblich erschweren. Wenn Sie nun die Idee haben, dass Sie die Fehlernummer gezielt auswerten können, sollten Sie diese Idee idealerweise gleich wieder vergessen. Einmal existieren Unmengen von Fehlernummern, zum anderen können Sie nicht davon ausgehen, dass eine neue Version der verwendeten Software (bei Datenbankverbindungen ist das z. B. ADO) dieselben Fehlernummern erzeugt. In ASP.NET existieren Fehlernummern erst gar nicht mehr. Dort werten Sie Klassen von Fehlern aus. Alle Fehler, die den Zugriff auf das Dateisystem betreffen, gehören z. B. der Klasse `IOException` an. Lassen Sie die Nummern also Nummern sein und machen Sie sich keinen Stress.

ASP.NET

Die Fehlerbehandlung unter ASP.NET ist komplett anders als unter ASP. Zunächst treten unter .NET im Allgemeinen keine Laufzeitfehler mehr auf, sondern »Ausnahmen« (Exceptions). Ausnahmen sind in Klassen organisiert. Die Klasse `Exception` ist die Basisklasse aller Ausnahmen. Von dieser Klasse sind eine Menge weitere Ausnahme-Klassen abgeleitet, die spezifische Ausnahmen behandeln. Die Klasse `DataException` ist z. B. für Ausnahmen beim Datenzugriff zuständig.

Die gezielte Fehlerbehandlung ist in ASP.Net wesentlich einfacher als in ASP. Anweisungen, bei denen in der Laufzeit ein Fehler auftreten kann, fügen Sie dazu in eine `Try-Catch`-Struktur ein. Im `Try`-Block bringen Sie die Anweisung(en) unter, die Fehler verursachen können. Im `Catch`-Block programmieren Sie die Fehlerbehandlung:

```
' Öffnen der Verbindung zu Datenbank 'Pubs'
' auf dem lokalen SQL Server
Dim con As System.Data.SqlClient.SqlConnection
con = New System.Data.SqlClient.SqlConnection( _
   "Server=(local);Database=Pubs;" & _
   "User ID=sa;Password=bandit")
Try
   con.Open()
Catch e As Exception
   ' Fehlerauswertung
  Response.Write("Beim Öffnen der Datenbank " & _
     "ist ein Fehler aufgetreten: " & _
     e.Message)
  Response.Write("<p>Benachrichtigen Sie den " & _
     "<a href='mailto:admin@panzerknacker.ag'> " & _
     "Webadiminstrator</a></p>")
End Try
```

Über die Variable e, die in der `Catch`-Anweisung angegeben ist, erhalten Sie Zugriff auf die Fehlerinformationen. Die Angabe der Klasse `Exception` bewirkt, dass dieser `Catch`-Block alle Fehlerklassen bearbeitet. Alternativ könnten Sie mehrere `Catch`-Blöcke einrichten, die separate Fehlerklassen bearbeiten. Die wichtigsten Eigenschaften der Klasse `Exception` sind `Message` (Text der Fehlermeldung) und `StackTrace` (enthält den Quellcode, an der der Fehler aufgetreten ist, und einige vorhergehende Anweisungen, die zur fehlerhaften Anweisung geführt haben). `StackTrace` wird zu Debugging-Zwecken verwendet. Damit können Sie bei unerwarteten Fehlern eine Information erhalten, an welcher Stelle im Quellcode der Fehler aufgetreten ist.

Ausnahmen, die Sie nicht speziell behandeln, können Sie über eine eigene ASPX-Seite anzeigen, die Sie in der `ErrorPage`-Eigenschaft der Seite angeben (falls Sie dies jetzt schon ausprobieren wollen: Beach-

ten Sie, dass die Option CustomErrors in der Datei Web.Config dazu mit dem Attribut "mode = On" gekennzeichnet sein muss, damit dies auch auf dem Server selbst funktioniert).

Alternativ können Sie auch das `Error`-Ereignis der Seite verwenden, um unbehandelte Fehler abzufangen. Fügen Sie dazu eine Ereignisprozedur im Head-Bereich der Seite ein:

```
<html>
<head><title>Fehlerdemo</title>

<script language="VB" runat="server">
 Private Sub Page_Error(ByVal sender As Object, _
  ByVal e As System.EventArgs)
  Response.Write("Fehler in " & Request.Url.ToString() & _
     ":" & Server.GetLastError.Message & _
     "<p>Stack-Trace: " & Server.GetLastError.StackTrace)
  Server.ClearError()
End Sub
</script>

</head>
...
```


7.7 Quellcode wieder verwenden: Include-Dateien

Ein großes Problem von ASP-Programmen ist, dass es zunächst nicht allzu einfach ist, Quellcode so zu verwalten, dass dieser in beliebig vielen ASP-Dokumenten wiederverwendet werden kann. In normalen Programmiersprachen stehen Ihnen zu diesem Zweck Module zur Verfügung (die auch als »Libraries« oder »Units« bezeichnet werden), über die Sie Ihre eigenen Funktionsbibliotheken aufbauen können. Da eine ASP-Datei prinzipiell alleine steht, sind solche Module in ASP 2/3 nicht möglich (in ASP.NET dagegen schon). Funktionsbibliotheken müssen Sie in ASP anders aufbauen.

Für wiederverwendbaren Programmcode können Sie mit einer COM-fähigen Programmiersprache COM-Komponenten erzeugen und diese in ASP einsetzen. Oft ist das aber zu viel Aufwand. Außerdem

erzeugen solche Komponenten auch Probleme: Wenn eine ASP-Seite eine Klasse aus einer COM-Komponente einmal instanziiert hat, gibt der IIS die DLL- oder EXE-Datei der Komponente nicht mehr so einfach frei. In den meisten Fällen muss der Webserver komplett neu gestartet werden, um die Komponente gegen eine neue Version austauschen zu können.

Die Alternative ist, ASP-Quellcode in separaten Dateien zu verwalten, die dann einfach in die ASP-Dokumente, in denen dieser Code benötigt wird, eingebunden werden. Dazu stehen Ihnen zwei Möglichkeiten zur Verfügung: Sie können Quellcode-Dateien erzeugen, die Sie mit einer Include-Anweisung statisch oder dynamisch in ASP-Dokumente einbinden. Die andere Möglichkeit ist, Quellcode-Dateien über `Server.Execute` aufzurufen.

7.7.1 Statische Include-Dateien

Include-Dateien, die statisch in ein ASP-Dokument eingebunden werden, sollten nur ASP-Code enthalten. Wenn Sie diese als Funktionsbibliothek verwenden (wie Sie gleich noch sehen werden, können Sie Include-Dateien noch für einen Trick verwenden), sollten diese nur Funktionen und eventuell Variablendeklarationen beinhalten. Sie können diese Dateien dann über

```
<!-- #include file="Dateiname" -->
oder
<!-- #include virtual="Dateiname" -->
```

in eine ASP-Datei einbinden. Der Unterschied zwischen `file` und `virtual` ist, dass Sie mit `file` eine normale Dateiangabe verwenden (relativ vom Pfad der ASP-Datei oder absolut vom Stammordner des IIS aus) und mit `virtual` eine von einem virtuellen Webordner ausgehende Dateiangabe. Wenn Sie Ihre Include-Dateien z. B. in einem speziellen Ordner speichern, auf den Sie mit einer normalen Dateiangabe nicht zugreifen können, erstellen Sie für diesen Ordner einen virtuellen Webordner und greifen dann über diesen auf die Include-Dateien zu.

Das folgende Beispiel verwendet eine Include-Datei »Tools.inc« mit zwei Prozeduren, die in demselben Ordner gespeichert ist wie die ASP-Dateien. Die Prozeduren sind in einen `script`-Tag eingebettet,

um mit ASP.NET kompatibel zu sein (ASP.NET erlaubt keine Prozeduren/Funktionen in ASP-Token):

```
<script language="VBScript" runat="server">
 Sub WriteLine(Text)
    Response.Write Text & "<br>"
 End Sub

 Sub WriteParagraph(Text)
    Response.Write "<p>" & Text & "</p>"
 End Sub
</script>
```

Diese Include-Datei wird dann im HTML-Bereich (!) der ASP-Datei eingebunden:

```
<%@Language="VBScript"%>
<%Option Explicit%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<title>Include-Datei</title>

<!-- Einbinden der Include-Datei -->
<!-- #include file="Tools.inc" -->

</head>
<body>

<%
 ' Anwender der Funktionen aus der Include-Datei
 WriteLine "Das ist eine Zeile, die mit der " & _
   "WriteLine-Funktion aus der Include-Datei " & _
   "geschrieben wurde."
%>

</body>
</html>
```

Besonders hilfreich sind Include-Dateien auch bei der Verwaltung von Anwendungsoptionen, die später eventuell angepasst werden müssen. Verwalten Sie diese Optionen in Form von Konstanten in einer Include-Datei, die in alle ASP-Dateien eingebunden wird, ist die Anpassung der Anwendung kein Problem.

Die ASP-Engine liest bei der Verarbeitung einer ASP-Datei neben der Datei selbst auch alle Include-Dateien ein und generiert daraus automatisch ein temporäres Dokument, das den kompletten Quellcode enthält. Dieses Dokument wird dann von der ASP-Engine ausgewertet. Beachten Sie, dass ASP-Code in Include-Dateien, der nicht innerhalb einer Funktion oder Prozedur gespeichert ist, genau wie normaler ASP-Code bei der Abarbeitung des ASP-Dokuments direkt mit abgearbeitet wird. Der folgende »Trick« nutzt diesen Umstand.

7.7.2 Include-Dateien dynamisch einbinden

Include-Dateien, die eigentlich ja statisch sind, können auch dynamisch eingebunden werden. Dieses Vorgehen ähnelt der Verwendung der `Execute`-Methode, die im nächsten Abschnitt beschrieben wird. Im Prinzip ist das ganz einfach. Wenn eine bestimmte Bedingung eintritt oder innerhalb einer Schleife binden Sie die Datei einfach ein. Das folgende Beispiel reagiert auf einen Laufzeitfehler beim Öffnen einer Datenbankverbindung damit, dass eine Include-Datei eingebunden wird:

```
' Öffnen der Verbindung zu Datenbank 'Pubs'
' auf dem lokalen SQL Server
Dim con
Set con = Server.CreateObject("ADODB.Connection")
On Error Resume Next
con.Open "Provider=SQLOLEDB;Data Source=(local);" & _
   "Initial Catalog=Pubs", "sa", ""
If Err.Number <> 0 Then
   %>
   <!-- #include file="Error.inc" -->
   <%
End If
```

Die Include-Datei beinhaltet lediglich die Ausgabe der Fehlermeldung:

```
<h1>Fehler aufgetreten</h1>

<%
 Response.Write Err.Description & "<br>"
%>

<p>Benachrichtigen Sie den
 <a href="admin@panzerknacker.ag">
 Webadiminstrator</a></p>
```

Wie Sie sehen, dürfen solche Include-Dateien auch HTML-Quellcode enthalten. Der Trick dabei ist, dass ASP für den Zeitpunkt des Einbindens kurz unterbrochen und danach wieder begonnen wird.

Wenn Sie an eine solche Include-Datei Argumente übergeben wollen, deklarieren Sie einfach in der ASP-Seite Variablen, schreiben die zu übergebenden Werte in diese Variablen und werten sie in der Include-Datei aus. Die folgende Include-Datei arbeitet z. B. mit einer Variablen für eine zusätzliche Info:


```
<h1>Fehler aufgetreten</h1>

<%
 ' Variablen ausgeben, die in der einbindenden
 ' ASP-Seite deklariert sind
 ' Info ausgeben
 Response.Write strInfo & "<br>"
 ' Fehlermeldung ausgeben
 Response.Write strError & "<br>"
%>

<p>Benachrichtigen Sie den
 <a href="admin@panzerknacker.ag">
 Webadiminstrator</a></p>
```

Die aufrufende ASP-Seite deklariert diese Variablen und Werte hinein:

```
...
<%
 ' Include-Datei dynamisch einbinden am Beispiel
 ' einer Fehlerbehandlung
 Dim fso, tst
 Set fso = _
    Server.CreateObject("Scripting.FilesystemObject")
 On Error Resume Next
 Set tst = fso.OpenTextFile("C:\BinNichtDa.txt")
 ' Fehler abfragen
 If Err.Number <> 0 Then
    ' Variablen für die Übergabe von Werten deklarieren
    Dim strInfo, strError
    ' und mit dem zu übergebenden Wert beschreiben
    strInfo = "Fehler beim Öffnen der Datei"
    strError = Err.Description
    ' Fehlerbehandlung wieder einschalten
    On Error Goto 0
    ' Include-Datei dynamisch einbinden
    %>
    <!-- #include file="Fehler.inc" -->
    <%
 End If
    ' Fehlerbehandlung wieder einschalten
    On Error Goto 0
%>
...
```

Alternativ können Sie natürlich auch Werte im `Session`-Objekt übergeben.

7.7.3 Dynamischer Aufruf von ASP-Seiten

ASP-Seiten können Sie ab dem IIS 5 über die `Execute`- oder `Transfer`-Methode des `Server`-Objekts dynamisch innerhalb anderer ASP-Seiten aufrufen. Die aufgerufene ASP-Seite kann HTML- und ASP-Quellcode beinhalten. Das folgende Beispiel illustriert, wie das funktioniert. Die Datei »Uhrzeit.inc« beinhaltet die Ausgabe der Uhrzeit mit HTML- und einmal als ASP-Code:

```
<h2>Uhrzeit</h2>
<b>Es ist genau <%=Time%> Uhr </b>
```

Eine ASP-Datei ruft diese Datei an geeigneter Stelle auf:

```
<%@Language="VBScript"%>
<%Option Explicit%>

<html>
<head>
<title>Include-Datei mit Execute aufrufen</title>

</head>
<body>

<h1>Include-Datei mit Execute aufrufen</h1>

<%
 ' Aufrufen der Include-Datei
 Server.Execute "Uhrzeit.inc"
%>

</body>
</html>
```

Das Ergebnis ist eine saubere HTML-Datei:

```
<html>
<head>
<title>Include-Datei mit Execute aufrufen</title>
</head>
<body>

<h1>Include-Datei mit Execute aufrufen</h1>

<h2>Es ist genau 16:48:30 Uhr </h2>

</body>
</html>
```

`Execute` und `Transfer` unterscheiden sich dadurch, dass ASP nach dem Aufruf von `Execute` die aufrufende Seite weiter abarbeitet, `Transfer` beendet die weitere Abarbeitung.

In ASP.NET (Beta 2, Build 9344) muss die mit `Execute` *oder* `Transfer` *aufgerufene Datei die Endung* .ASPX *besitzen. Ansonsten führt das .NET-Framework den enthaltenen ASP-Code nicht aus.*

Entgegen den Microsoft-Beispielen in der IIS-Hilfe und im Internet sollte die eingebundene Datei meiner Meinung nach keine komplette HTML-Deklaration enthalten. Die ASP-Engine bindet Include-Dateien immer komplett ein. Wenn die Include-Datei die Basis-HTML-Tags beinhaltet, erhalten Sie diese gleich mehrfach im Ergebnisdokument. Auch wenn der Internet Explorer vielleicht damit zurechtkommt, müssen andere Browser so erzeugte Seiten noch lange nicht korrekt darstellen.

ASP funktioniert eigentlich im Großen und Ganzen sehr gut. Mit der `Execute`- und der `Transfer`- Methode habe ich aber endlich etwas, worüber ich nörgeln kann: Ein Problem ist die Übergabe von Argumenten an so aufgerufene ASP-Dateien. Microsoft behauptet, man könne damit »eine komplexe Anwendung in einzelne Module« aufteilen. Dazu muss man aber auch Argumente übergeben können. URL-Argumente sind, ebenfalls entgegen der IIS-Hilfe (»Der `Pfad`-Parameter kann eine Abfragezeichenfolge enthalten«) nicht möglich. Ein Versuch, URL-Argumente zu übergeben, resultierte in dem Laufzeitfehler »Es wurde eine ungültige URL-Form oder ein voll gekennzeichneter absoluter URL verwendet. Verwenden Sie relative URLs«. Anders als bei mit #Include eingebundenen Dateien können in der Include-Datei leider auch keine Variablen verwendet werden, die in der aufrufenden Datei deklariert wurden. Zur Übergabe von Argumenten bleibt dann nur noch das `Session`-Objekt.

8 Arbeiten mit Objekten

In ASP arbeiten Sie sehr häufig mit Objekten. In den Beispielen der vorhergehenden Kapitel habe ich immer wieder das `Response`-Objekt verwendet, um mit dessen `Write`-Methode HTML-Quellcode zu erzeugen. Dieses Kapitel beschreibt deshalb zunächst die integrierten ASP-Objekte (die auch in ASP.NET verwendet werden). Da das nächste Kapitel wichtige ASP-Programmiertechniken beschreibt und dabei intensiv die integrierten ASP-Objekte nutzt, beschreibe ich nur kurz die Eigenschaften, Methoden und Ereignisse dieser Objekte und verzichte auf weiterführende Beispiele.

Andere Objekte, die häufig in ASP eingesetzt werden, sind in separaten COM-Komponenten gespeichert. Wichtige COM-Komponenten wie ADO (zum Datenbankzugriff) und die Scripting Runtime-Bibliothek (mit Klassen zum Dateizugriff und für Auflistungen) werden automatisch mit dem IIS installiert. Dieses Kapitel zeigt, wie Sie Objekte aus solchen Komponenten instanziieren und verwenden. Da ADO im nächsten Kapitel intensiver besprochen wird, verwende ich eine sehr häufig eingesetzte Klasse der Scripting Runtime-Bibliothek, die `Dictionary`-Klasse, die zur Erzeugung von Auflistungen eingesetzt wird.

Ein wichtiges Thema ist auch das Programmieren von eigenen COM-Komponenten, die dann in ASP eingesetzt werden. Da solche Komponenten mit einer richtigen Programmiersprache entwickelt werden, haben Sie bei der Programmierung wesentlich mehr Möglichkeiten als mit ASP alleine. Die so erzeugten Klassen sind kompiliert, weswegen die Performance von ASP-Seiten erheblich beschleunigt wird. Mehr dazu finden Sie im entsprechenden Artikel auf der Webseite.

8.1 Übersicht über die vordefinierten Objekte

8.1.1 Die ASP-Objekte

ASP enthält mittlerweile sieben integrierte Objekte, die in Tabelle 8.1 aufgelistet und ab Seite 209 näher beschrieben werden. Bis auf das `ASPError`- und das `ObjectContext`-Objekt sind diese Objekte bereits erzeugt. Das `ASPError`- und das `ObjectContext`-Objekt werden von speziellen Funktionen zurückgegeben. Sie müssen die ASP-Objekte also nicht selbst erzeugen.

Objekt	Bedeutung
`Application`	Über das `Application`-Objekt können Sie Daten global über die gesamte Anwendung global speichern und verwalten.
`Session`	Über das `Session`-Objekt speichern Sie Daten global über eine Sitzung und können einige Einstellungen der Sitzung beeinflussen.
`Server`	Über dieses Objekt erhalten Sie Zugriff auf einige Methoden und Eigenschaften des IIS. Die Eigenschaft `ScriptTimeout` legt fest, wie lange ein ASP-Skript maximal laufen darf.
`Request`	Mit dem `Request`-Objekt können Sie Daten abfragen, die der Client im HTTP-Header gesendet hat. Damit fragen Sie z. B. URL-Argumente oder Formulareingaben ab.
`Response`	Über dieses Objekt können Sie Daten zum Client senden. Die primäre Aufgabe dieses Objekts ist die Ausgabe von Daten im HTML-Dokument.
`ASPError` (ab IIS 5)	Das `ASPError`-Objekt wird von der `GetLastError`-Methode des `Server`-Objekts zurückgegeben, über das Sie Informationen zu aufgetretenen Fehlern abrufen können.
`ObjectContext`	Dieses Objekt wird benötigt, wenn die ASP-Seite im MTS oder in den Komponentendiensten von COM+ ausgeführt wird. Mit `ObjectContext` steuern Sie in ASP die Ausführung von Transaktionen.

Tabelle 8.1: Die integrierten ASP-Objekte

8.1.2 Die integrierbaren COM-Komponenten

Bei der Installation des IIS werden einige COM-Komponenten mit installiert, deren Klassen Sie in ASP-Dokumenten verwenden können (wie ich es ab Seite 221 beschreibe). Tabelle 8.2 listet alle unter dem IIS 5 verfügbaren Klassen auf. Einige davon sind eigentlich vollkommen uninteressant, weil Sie deren einfache Funktion meist sehr leicht selbst nachbilden können. Beachten Sie, dass einige der Klassen der MSWC-Bibliothek nicht unter dem IIS 4 verfügbar sind.

Klasse	Bedeutung

`ADODB.Connection`
`ADODB.Recordset`
`ADODB.Command` u. A.
	Die Klassen der ActiveX Data Objects-Bibliothek (die nicht nur mit dem IIS, sondern von fast allen Microsoft-Anwendungen installiert werden) werden für den Zugriff auf Datenbanken und andere Datenquellen benötigt (siehe Kapitel 10).

`MSWC.AdRotator`	Über diese Klasse können Sie Werbung auf einer Seite so anzeigen, dass diese in regelmäßigen Abständen ausgetauscht wird.
`MSWC.BrowserType`	Diese Klasse ermöglicht Ihnen, herauszufinden, welche Fähigkeiten der Browser des Benutzers besitzt. Sie können z. B. ermitteln, ob der Browser VBScript unterstützt.
`MSWC.ContentRotator`	Mit dieser Klasse können Sie den Inhalt einer HTML-Seite bei jedem Besuch der Seite nach einem Plan dynamisch austauschen.
`MSWC.Counters`	`Counters` ermöglicht die Verwaltung von beliebig vielen Zählern, z. B. für Seitenzugriffe. Über die `Increment`-Methode können Sie den gespeicherten Wert um 1 addieren. Super.
`MSWC.IISLog`	Über diese Klasse können Sie Einträge auf den Protokolldateien des IIS auslesen. Ein Anwendungsprotokoll ist z. B. wichtig, um herauszufinden, welche Benutzer wann welche Seiten abgefragt haben.

Klasse	Bedeutung
MSWC.MyInfo	In einem Objekt dieser Klasse können Sie Informationen über den Webadministrator speichern, wie z. B. dessen Adresse, Telefonnummer und E-Mail-Adresse. Die Daten werden automatisch in einer XML-Datei gespeichert und bleiben deswegen bestehen (wenigstens so lange, bis die XML-Datei gelöscht oder der Rechner neu installiert wird).
MSWC.NextLink	Diese Klasse verwaltet eine Liste von URLs. Mit den Methoden und Eigenschaften können Sie mehrere Webseiten so verknüpfen, dass diese wie ein Buch verwendet werden können. Sie können dem Benutzer z. B. eine Möglichkeit geben, die nächste und die vorherige Seite anzuzeigen, und können ein Inhaltsverzeichnis generieren.
MSWC.PageCounter	Diese Klasse ermöglicht die Implementierung eines einfachen Seitenzählers, der die »Hits« automatisch in einer Datei verwaltet (deren Dateiname in der Registry eingestellt werden kann).
MSWC.Permission-Checker	Diese interessantere Klasse der MSWC-Bibliothek ermöglicht die Überprüfung, ob der aktuelle Webbenutzer in einem geschützten Web Zugriffsrechte auf eine bestimmte ASP-Datei besitzt.
MSWC.Status	Über diese Klasse erhalten Sie Informationen zum Serverstatus. Sie können z. B. die Anzahl der Besuche seit dem letzten Start oder die Anzahl der Zugriffe seit Mitternacht abfragen.
MSWC.Tools	Diese Klasse bietet einige wenige Hilfsmethoden, wie z. B. FileExists, über die Sie ermitteln können, ob eine bestimmte Datei existiert.

Klasse	Bedeutung
Scripting.File-SystemObject	Die FileSystemObject-Klasse (die wie ADODB nicht nur vom IIS installiert wird) gibt Ihnen Zugriff auf das Dateisystem des Servers. Damit können Sie z. B. Textdateien lesen und schreiben und Ordner anlegen.

Tabelle 8.2: Die Klassen der mit dem IIS installierten COM-Komponenten

8.2 Die integrierten ASP-Objekte

8.2.1 Das Application-Objekt

Das Application-Objekt besitzt die in Tabelle 8.3 aufgelisteten Eigenschaften und Methoden und einige Ereignisse (Tabelle 8.4).

Eigenschaft / Methode	Beschreibung
Contents	Contents ist eine Auflistung, in der Sie beliebige Daten applikationsweit speichern können. Die hier gespeicherten Daten können innerhalb einer beliebigen Sitzung von allen ASP-Seiten der Anwendung gelesen und beschrieben werden. Mit der Item-Methode können Sie Elemente anfügen: Application.Contents.Item(*Schlüssel*) = Wert. Über diesen Schlüssel können Sie dann, wieder über Item, auf die gespeicherten Elemente zugreifen. Mit der Remove-Methode löschen Sie einzelne Elemente über deren Schlüssel oder einen Integer-Index (der die Position des Elements innerhalb der Auflistung angibt), RemoveAll löscht alle gespeicherten Elemente. In ASP.NET wird Contents aus Kompatibilitätsgründen zwar unterstützt. Die Eigenschaften und Methoden dieser Auflistung sind in ASP.Net aber auch direkt dem Application-Objekt zugeordnet. Statt Application.Contents.Item schreiben Sie also besser Application.Item. Außerdem kennt das Application-Objekt in ASP.NET zusätzliche Elemente wie die Eigenschaft AllKeys, die einen String mit allen Schlüsseln zurück gibt.

Eigenschaft / Methode	Beschreibung
StaticObjects	Diese Auflistung verweist auf die Objekte, die der Anwendung in der Datei GLOBAL.ASA (ASP) bzw. GLOBAL.ASAX (ASP.NET) über einen object-Tag hinzugefügt wurden und den Gültigkeitsbereich »Application« besitzen (siehe Seite), also applikationsweit gültig sind.
Lock	Mit Lock sperren Sie den Zugriff auf die Contents-Auflistung für den Zeitraum, in dem Sie selbst in diese Auflistung schreiben. Damit vermeiden Sie Probleme, die entstehen könnten, wenn mehrere Benutzer gleichzeitig dieselben Elemente der Contents-Auflistung beschreiben.
Unlock	Unlock entsperrt den Zugriff auf Contents wieder.

Tabelle 8.3: Die Eigenschaften und Methoden des Application-Objekts (bei ASP.NET nur die wichtigsten)

Ereignis	Beschreibung
OnStart (ASP) Start (ASP.NET)	Dieses Ereignis wird aufgerufen, wenn die Anwendung startet.
OnEnd (ASP) End (ASP.NET)	Dieses Ereignis wird aufgerufen, wenn die Anwendung beendet wird.
BeginRequest	Dieses ASP.NET-Ereignis wird aufgerufen, wenn ein Client eine Anforderung an die Anwendung sendet.
Authenticate-Request	Dieses ebenfalls nur in ASP.NET verfügbare Ereignis wird aufgerufen, wenn ein Client für einen geschützten Webordner Authentifizierungsinformationen übergibt um den Zugriff zu authentifizieren.
Error	Dieses ASP.NET-Ereignis wird ausgelöst, wenn ein Fehler auftritt.
EndRequest	Dieses ASP.NET-Ereignis wird aufgerufen, wenn die Abarbeitung einer Anforderung beendet ist.

Tabelle 8.4: Die Ereignisse des Application-Objekts (bei ASP.Net nur die wichtigsten)

Wenn Sie die Ereignisse des Application-Objekts (und auch die des Session-Objekts) auswerten wollen, müssen Sie in einer Datei mit dem Namen Global.asa eine Ereignisprozedur deklarieren. Wie das geht, zeige ich ab Seite 219.

8.2.2 Das Session-Objekt

Über das Session-Objekt erhalten Sie Zugriff auf die Sitzung, in der die aktuelle ASP-Seite ausgeführt wird. Unter anderem können Sie im Session-Objekt wie im Application-Objekt Daten speichern, die dann allerdings nur für diese eine Sitzung global sind. Diese Technik wird häufig genutzt, um Daten über mehrere ASP-Dokumente global zu speichern.

Wie das Application-Objekt kennt das Session-Objekt die Ereignisse OnStart und OnEnd. OnStart wird allerdings beim Session-Objekt für jeden Client separat aufgerufen, wenn dieser eine Verbindung zu einer ASP-Anwendung aufbaut, für die er noch keine Sitzung besitzt. OnEnd wird dann aufgerufen, wenn das Sitzungs-Timeout abgelaufen ist oder der Webserver heruntergefahren wird.

Eigenschaft / Methode	Beschreibung
Contents	In Contents können Sie wie beim Application-Objekt Daten speichern, die dann allerdings nur sitzungsweit verfügbar sind.
StaticObjects	Diese Auflistung verweist auf die Objekte, die der Anwendung in der Datei GLOBAL.ASA über einen object-Tag hinzugefügt wurden und den Gültigkeitsbereich »Sitzung« besitzen (siehe Seite 224), also sitzungsweit gültig sind.
CodePage	In dieser Eigenschaft können Sie die Zeichentabelle (Codepage), die ein ASP-Dokument standardmäßig verwendet, vorübergehend für einzelne Aktionen ändern, um z. B. einen Text einzulesen, der mit einer anderen Zeichentabelle erzeugt wurde.
LCID	Über LCID können Sie die Standard-Ländercode-ID (die über das ASP-Attribut @ LCID eingestellt wird) für einzelne Aktionen ändern.

Eigenschaft / Methode	Beschreibung
LCID *(Fortsetzung)*	Über die Ländercode-ID ermittelt der IIS die länderspezifischen Einstellungen für Zahlen und Datumswerte. Diese Einstellungen werden beim impliziten oder expliziten Formatieren von Zahl- oder Datumswerten (z. B. mit FormatDateTime) innerhalb des Script-Programms verwendet.
IsNewSession	Diese nur in ASP.NET verfügbare Eigenschaft gibt eine Information darüber, ob die aktuelle Anforderung die Sitzung erzeugt hat. Damit können Sie darauf reagieren, wenn der Benutzer bei der Arbeit mit Ihrer Anwendung zwischenzeitlich zu lange gewartet hat, so dass die Sitzung beendet wurde.
SessionID	SessionID gibt lesenden Zugriff auf die ID der Sitzung. Diese Information ist wenig hilfreich, da Sitzungs-IDs dynamisch vergeben werden und spätere Sitzungen dieselbe ID besitzen können wie ältere.
Timeout	In dieser Eigenschaft können Sie das voreingestellte Timeout der Sitzung im Programm für jede Sitzung separat einstellen (in Minuten). Dies kann z. B. dann notwendig sein, wenn bestimmte Teile der Anwendung vom Benutzer länger bearbeitet werden, als der normale Sitzungs-Timeout erlaubt. In diesem Fall sollten Sie den Timeout der Sitzung für jede ASP-Seite separat und individuell über Session.Timeout einstellen.
Abandon	Diese Methode zerstört das Session-Objekt und damit alle gespeicherten Daten. Abandon sollten Sie immer dann aufrufen, wenn Sie die im Session-Objekt gespeicherten Daten nicht mehr länger benötigen (um Ressourcen zu sparen). Die Sitzung wird allerdings nicht gelöscht, die Sitzungs-Id bleibt bestehen. Der Aufruf dieser Methode führt aber dazu, dass das OnEnd-Ereignis des Session-Objekts ausgeführt wird. Wenn Sie danach wieder auf das Session-Objekt zugreifen, wird auch das OnStart-Ereignis erneut aufgerufen.

Tabelle 8.5: Die Eigenschaften und Methoden des Session-Objekts (bei ASP.NET nur die wichtigsten)

Ereignis	Beschreibung
`OnStart` (ASP) `Start` (ASP.NET)	wird aufgerufen, wenn die Sitzung startet. In der verbundenen Ereignisprozedur können Sie die Sitzung initialisieren.
`OnEnd` (ASP) `End` (ASP.NET)	wird aufgerufen, wenn die Sitzung beendet wird. In der Ereignisprozedur können Sie Ressourcen freigeben, die im Start-Ereignis reserviert wurden.

Tabelle 8.6: Die Ereignisse des Session-Objekts

8.2.3 Das Server-Objekt

Über das `Server`-Objekt erhalten Sie Zugriff auf Methoden und Eigenschaften des Webservers. Wichtige Methoden sind z. B. `HTMLEncode` und `UrlEncode` zur Kodierung beliebiger Zeichenketten in deren HTML- bzw. URL-Entsprechung.

Eigenschaft / Methode	Beschreibung
`ScriptTimeout`	Über diese Eigenschaft können Sie das voreingestellte Timeout für die Ausführung von ASP-Seiten für Ihre Anwendung verändern. In ASP-Seiten, die sehr viel Programmcode enthalten, kann das voreingestellte Timeout schon einmal zu kurz sein. Merken Sie sich das originale Timeout dann am Start der Seite in einer Variablen, setzen Sie das neue Timeout und das alte am Ende der Seite wieder zurück.
`MachineName`	Diese nur unter ASP.NET verfügbare Eigenschaft gibt den Namen des Rechners zurück.
`CreateObject` (*ProgId*)	Diese Methode wird verwendet, um aus Klassen, die in COM-Komponenten gespeichert sind, Objekte zu erzeugen. Wie das geht, zeige ich ab Seite 223.

Eigenschaft / Methode	Beschreibung
`Execute(`*Pfad*`)`	Diese Methode, die ab dem IIS 5 verfügbar ist, führt eine ASP-Datei so aus, als wäre sie Teil des aufrufenden ASP-Skripts. Damit können Sie wiederverwendbaren Code in ASP-Dateien speichern und bei Bedarf aufrufen. `Execute` ist eine Alternative zum Einbinden von ASP-Quellcode-Dateien mit `#include`.
`GetLastError`	Der IIS ruft beim Eintritt eines unbehandelten Laufzeitfehlers im Skript einer ASP-Seite ein in der Konfiguration des Webordners eingestelltes Fehlerbehandlungs-ASP-Dokument auf, das Informationen zum Fehler ausgibt. In diesem Dokument wird `GetLastError` verwendet, um Informationen zum aufgetretenen Fehler zu ermitteln. `GetLastError` gibt ein `ASPError`-Objekt zurück. Diese Methode ist erst ab dem IIS 5 verfügbar. Die Behandlung von Laufzeitfehlern wird in Kapitel 7 beschrieben.
`HTMLEncode(` *Zeichenfolge*`)`	Diese Methode kodiert den übergebenen Text so, dass dieser HTML-konform ist. Das Zeichen > wird in die HTML-Zeichenfolge > konvertiert. `HTMLEncode` sollten Sie immer dann verwenden, wenn Sie Texte ausgeben, die HTML-Sonderzeichen enthalten oder enthalten könnten. Wenn Sie z. B. Daten aus einer Datenbank auslesen und ausgeben, könnte es immer sein, dass in diesen HTML-Sonderzeichen enthalten sind. Geben Sie diese Daten dann unkonvertiert aus, stellt der Browser das Dokument nicht korrekt dar.
`MapPath(`*Pfad*`)`	Die `MapPath`-Methode setzt eine relative oder virtuelle Pfadangabe in einen physischen Pfad um. Sie können bei der Pfadangabe vom Ordner der Webanwendung ausgehen, indem Sie nichts weiter angeben (`MapPath("`*Pfad*`")` hängt den angegebenen Pfad an den Pfad des Webordners an). Wenn Sie vom Stammordner des IIS ausgehen wollen, beginnen Sie die Pfadangabe mit einem Slash (`MapPath("/`*Pfad*`")` hängt den angegebenen Pfad an den Stammordner des IIS an).

Eigenschaft / Methode	Beschreibung
`MapPath`(*Pfad*) *(Fortsetzung)*	Alternativ können Sie auch den Punkt in der Pfadangabe verwenden (`MapPath(".")` ergibt das Verzeichnis des Webordners, `MapPath("..")` ergibt den dem Webordner übergeordneten Ordner). Diese Methode benötigen Sie immer dann, wenn Sie im ASP-Skript auf Dateien oder Ordner zugreifen wollen, die relativ zum IIS-Stammordner oder relativ zum Webordner gespeichert sind.
`Transfer`(*Pfad*)	Diese ab dem IIS 5 verfügbare Methode beendet die Ausführung des aktuellen ASP-Dokuments und fügt das angegebene Dokument an der Stelle im HTML-Code ein, an der diese Methode ausgeführt wird. Mit dieser Methode können Sie z. B. einen erforderlichen Login innerhalb einer ASP-Seite anzeigen. Die andere Asp-Seite darf auch innerhalb eines anderen Webordners gespeichert sein.
`UrlEncode` (*Zeichenfolge*)	`URLEncode` kodiert den übergebenen String so, dass dieser den Regeln für URLs entspricht. In einer korrekten URL müssen Leerzeichen durch ein + und Sonderzeichen durch deren hexadezimalen ANSI-Wert dargestellt werden (der Bindestrich wird z. B. zu »%2D«). Diese Methode benötigen Sie immer dann, wenn Sie in der URL Argumente übergeben.

Tabelle 8.7: Die Eigenschaften und Methoden des Server-Objekts (bei ASP.NET nur die wichtigsten)

8.2.4 Das Request-Objekt

Über das Request-Objekt erhalten Sie Zugriff auf die Daten, die der Client bei der Anforderung der aktuellen ASP-Seite im HTTP-Header und in der URL zum Webserver gesendet hat. Dazu gehören u. a. Informationen zum verwendeten Browser, Daten, die ein Benutzer in einem Formular eingegeben hat, und URL-Argumente.

Eigenschaft / Methode	Beschreibung
`ClientCertificate` (*Schlüssel*[*Unterfeld*])	Über diese Auflistung erhalten Sie Zugriff auf das Zertifikat des Client bei einer geschützten (SSL-)Verbindung.
`Cookies`	Die `Cookies`-Auflistung speichert alle Cookies, die die Anwendung auf dem Client-Rechner gespeichert hat. Cookies werden in Kapitel 9 behandelt.
`Form`	Die `Form`-Auflistung speichert die Daten aller Eingabefelder eines Formulars, das der Anwender abgesendet hat. Bei der Auswertung von Formulareingaben greifen Sie auf diese Auflistung zurück. In Kapitel 9 finden Sie Informationen darüber, wie das programmiert wird.
`QueryString`	In dieser Auflistung speichert ASP alle in der URL übergebenen Argumente.
`ServerVariables`	Über diese Auflistung erhalten Sie Zugriff auf verschiedene Umgebungsvariablen des Servers, aber auch auf Informationen über den anfordernden Client. In einem geschützten Web können Sie über den Schlüssel "LOGON_USER" z. B. erfahren, welcher Windows-Benutzer sich eingeloggt hat. Der Schlüssel "REMOTE_ADDR" ergibt die IP-Adresse des Client, "HTTP_USER_AGENT" ergibt Informationen zum Browser. In der IIS-Hilfe zum `Request`-Objekt sind die verfügbaren Schlüssel recht gut erläutert.
`Browser`	Gibt in ASP.NET ein `HttpBrowserCapabilities`-Objekt zurück, über das Sie die verschiedenen Möglichkeiten des Browsers abfragen können. Die Eigenschaft `JavaScript` dieses Objekts sagt aus, ob der Browser JavaScript unterstützt.

Eigenschaft / Methode	Beschreibung
`TotalBytes`	Diese Eigenschaft gibt die Größe der in der HTTP-Anforderung gesendeten Daten zurück. Sie benötigen diese Methode, wenn Sie binäre Daten (Dateien) auswerten wollen, die der Client zum Webserver gesendet hat (was aber auch einfacher geht, wie ich im Online-Artikel »ASP-Tipps und Tricks« beschreibe).
`BinaryRead(`*Zahl*`)`	Diese Methode liest die binären Daten, die der Client zum Server gesendet hat, um eine Datei zu übertragen.

Tabelle 8.8: Die Eigenschaften und Methoden des Request-Objekts (wie immer bei ASP.NET nur die wichtigsten)

8.2.5 Das Response-Objekt

Aufgabe des `Response`-Objekts ist, Daten in die Antwort zum Client zu schreiben. Die Ihnen bereits bekannte `Write`-Methode schreibt HTML-Quelltext. Sie können über dieses Objekt aber auch den HTTP-Header beschreiben und so z. B. Cookies zum Client senden.

Eigenschaft	Beschreibung
`Cookies`	Über diese Auflistung können Sie Cookies zum Client senden. Cookies werden im nächsten Kapitel behandelt.
`Buffer`	Im IIS 4 werden alle HTML-Ausgaben per Voreinstellung direkt zum Client gesendet. Mit `Response.Buffer = True` können Sie erreichen, dass der IIS alle Ausgaben zunächst puffert und erst dann sendet, wenn Sie `Response.Flush` aufrufen. Im IIS 5 ist der Puffer per Voreinstellung immer eingeschaltet. Puffer ermöglichen, dass Sie auch mitten in einer ASP-Seite mit `Response.Redirect` zu einem anderen Dokument verzweigen können.

Eigenschaft	Beschreibung
`Expires`	Über diese Eigenschaft können Sie festlegen, nach wie vielen Minuten eine Seite abläuft. Ein Browser, der abgerufene Seiten cached, fordert diese erst dann neu an, wenn sie abgelaufen sind. Mit `Response.Expires = 0` können Sie erreichen, dass die Seite sofort abläuft. Damit vermeiden Sie viele Probleme bei hochdynamischen ASP-Seiten.
`ExpiresAbsolute`	Mit dieser Eigenschaft können Sie für den Ablauf der Seite ein Datum mit Uhrzeit angeben.
`Status`	Über diese Eigenschaft können Sie den HTTP-Antwortstatus selbst setzen. Wenn Sie z. B. eine eigene Benutzerverwaltung implementiert haben und ein Benutzer loggt sich nicht gültig ein, senden Sie den Status »401 Unauthorized«, wenn Sie keine eigene Fehlermeldung ausgeben wollen.

Tabelle 8.9: Die wichtigsten Eigenschaften des Response-Objekts (dieses Mal auch nur die wichtigsten von ASP)

Methode	Beschreibung
`Clear`	`Clear` löscht alle gepufferten Daten, wenn der Puffer eingeschaltet ist. Diese Methode sollten Sie immer aufrufen, wenn die ASP-Seite gepuffert wird und Sie mit `End` beenden oder mit `Redirect` zu einer anderen Seite verzweigen.
`Flush`	`Flush` führt dazu, dass gepufferte Daten sofort ausgegeben werden. Diese Methode wird implizit am Ende der Seite aufgerufen, wenn die Pufferung eingeschaltet ist.
`End`	Mit dieser Methode können Sie die Ausführung einer Seite beenden. Wenn ein Fehler aufgetreten ist und Sie melden diesen Fehler, beenden Sie danach mit `End` die weitere Ausführung.

Methode	Beschreibung
`Redirect(`*Url*`)`	Mit `Redirect` können Sie eine Anforderung auf eine andere Seite umleiten. Im IIS 4 müssen Sie beachten, dass dazu eigentlich immer die Pufferung im Programm eingeschaltet sein muss. Eine Umleitung funktioniert nur dann, wenn noch keine HTTP-Header zum Client gesendet wurden.
`Write(`*Text*`)`	Die `Write`-Methode erzeugt HTML-Quellcode.

Tabelle 8.10: Die wichtigsten Methoden des Response-Objekts

8.2.6 Application- und Session-Ereignisse auswerten: Die Datei Global.asa(x)

Um die Ereignisse des Application- und des Session-Objekts auswerten zu können, müssen Sie passende Ereignisprozeduren deklarieren. Diese müssen in ASP in einer Datei GLOBAL.ASA und in ASP.NET in einer Datei GLOBAL.ASAX deklariert werden. Diese Dateien müssen im Stammordner der Anwendung gespeichert sein.

In ASP sieht die Grundstruktur dieser Datei folgendermaßen aus:

```
<script language="VBScript" runat="Server">
 Sub Application_OnStart
  ' Auswertung des Starts der Anwendung
 End Sub

 Sub Application_OnEnd
  ' Auswertung des Endes der Anwendung
 End Sub

 Sub Session_OnStart
  ' Auswertung des Starts einer Sitzung
 End Sub

 Sub Session_OnEnd
  ' Auswertung des Endes einer Sitzung
 End Sub
</script>
```

In ASP.NET können Sie dieselbe Grundstruktur verwenden. Beachten Sie, dass die Namen der Ereignisse nicht mit »On« beginnen.

Auf den Start der Anwendung brauchen Sie in der Praxis eigentlich nie zu reagieren. Die Beispiele, die ich bisher gesehen habe, haben im Start-Ereignis der Anwendung immer nur einfache Konfigurationsdaten in Application-Variablen geschrieben. Da Sie dafür auch Konstanten in Include-Dateien verwenden können, sehe ich nicht viel Sinn in diesem Ereigniss.

Den Start und das Ende einer Sitzung müssen Sie schon öfter auswerten. Beim Start initialisieren Sie die Sitzung, beim Beenden geben Sie Ressourcen frei, die Sie beim Start reserviert haben.

Manchmal ist es sinnvoll, Objekte beim Sitzungsstart zu erzeugen, damit diese während der gesamten Sitzung zur Verfügung stehen. Um zum Beispiel eine permanente Datenbankverbindung über die gesamte Sitzung zur Verfügung zu haben, müssen Sie diese im Start-Ereignis erzeugen und öffnen und im Ende-Ereignis wieder schließen:

```
Sub Session_OnStart
 ' Datenbankverbindung erzeugen und öffnen
 Set  Session("con") = _
    Server.CreateObject("ADODB.Connection")
 Session("con").Open "Provider=SQLOLEDB;" & _
    "Data Source=Zaphod;Initial Catalog=Orders"
End Sub

Sub Session_OnEnd
 ' Datenbankverbindung wieder schließen
 Session("con").Close
End Sub
```

Das Erzeugen von Objekten im Start-Ereignis der Sitzung wird häufig nicht allzu gerne gesehen. Der Server wird schließlich für die gesamte Dauer der Sitzung mit den reservierten Ressourcen belastet. Wenn es prinzipiell möglich ist, Objekte nur dann zu erzeugen, wenn Sie diese auch tatsächlich benötigen, und danach wieder freizugeben, sollten Sie dies auch so machen. In seltenen Fällen, z. B. dann wenn Sie dem Benutzer die Möglichkeit geben wollen, durch eine geöffnete Datenbanktabelle zu browsen, müssen Sie die betroffenen Objekte aber sitzungsweit verwalten.

8.3 Arbeiten mit externen Klassen

ASP-Skripte sind in der Lage, beliebige Objekte aus COM-Komponenten zu instanzieren, die auf dem System gespeichert sind. Beachten Sie, dass Ihnen unter ASP.NET in den Basisklassen des .NET-Framework meist wesentlich flexiblere Möglichkeiten zur Verfügung stehen. In ASP.NET können Sie also meist auf die COM-Komponenten des Systems verzichten.

8.3.1 Objekte aus COM-Komponenten erzeugen

Objekte, deren Klassen in COM-Komponenten gespeichert sind, können Sie in ASP einmal im Quellcode über die `CreateObject`-Methode des `Server`-Objekts erzeugen. Eine andere Möglichkeit ist, diese Objekte mit einem speziellen `object`-Tag zu erzeugen.

Erzeugen mit CreateObject

Der `CreateObject`-Methode des `Server`-Objekts übergeben Sie die »ProgId« der Klasse, die Sie instanzieren wollen. Diese setzt sich zusammen aus dem Namen der Komponente und dem Namen der Klasse:

```
Dim o
Set o = Server.CreateObject("Komponentenname.Klasse")
```

Beachten Sie, dass Sie Objekte in VBScript immer mit `Set` zuweisen müssen. Der `CreateObject`-Methode übergeben Sie den Namen der Komponente und den Namen der Klasse (vgl. Seite 207). Ein Browsertype-Objekt erzeugen Sie so:

```
Dim bt
Set bt = Server.CreateObject("MSWC.BrowserType")
```

Falls die Komponente oder die Klasse nicht auf dem Server registriert ist, resultiert ein Laufzeitfehler »Ungültige ProgID«.

Ist das Objekt erzeugt, können Sie das Objekt wie jedes andere Objekt benutzen:

```
Response.Write "Browser: " & bt.Browser & "<br>"
Response.Write "Version: " & bt.Version & "<br>"
```

Beachten Sie, dass das `BrowserType`-Objekt in diesem Beispiel den Internet Explorer 5.x interessanterweise als Netscape 4.0 erkennt.

Objekte mit dem object-Tag erzeugen

Alternativ zu CreateObject können Sie Objekte (im HTML-Bereich der Seite!) auch mit dem `object`-Tag erzeugen:

```
<object id="Id" progid="ProgID" runat="Server">
</object>
```

Der folgende Quelltext erzeugt ein Tools-Objekt:

```
<object id="tool" progid="MSWC.Tools" runat="Server">
</object>
```

Über die Id des Objekts können Sie im ASP-Skript auf das Objekt zugreifen:

```
<%
 Response.Write "Zufallszahl: " & tool.Random & "<br>"
%>
```

Objekte, die so auf ASP-Seiten erzeugt werden, gelten nur auf der jeweiligen Seite. Sie werden erzeugt, wenn die ASP-Engine bei der Abarbeitung der Seite an der Stelle der Deklaration angekommen ist, und zerstört, wenn die Abarbeitung beendet ist.

Wenn Sie die Deklaration in der Datei GLOBAL.ASA(X) vornehmen, können Sie ein statisches Objekt auch sitzungs- oder anwendungsweit erzeugen. Das zusätzliche Attribut `scope` legt dabei den Gültigkeitsbereich mit den Werten "application" oder "session" fest. Ein Tool-Objekt, das sitzungsweit gelten soll, erzeugen Sie so:

```
<object id="tool" progid="MSWC.Tools" runat="Server"
 scope="Session">
</object>
```

Viel Sinn sehe ich im `object`-Tag allerdings nicht.

8.3.2 Auflistungen mit der Dictionary-Klasse

Die `Dictionary`-Klasse, die Bestandteil der Scripting-Bibliothek ist, ist unter ASP eine der wichtigsten COM-Klassen. Mit Objekten dieser Klasse können Sie sehr flexible Auflistungen erzeugen. Beachten Sie,

dass Sie unter ASP.NET auch verschiedene Auflistungsklassen aus der Klassenbibliothek des .NET-Framework verwenden können.

Wie der Name schon sagt, eignet sich ein `Dictionary`-Objekt nicht nur zur Speicherung von Objekten, sondern auch zur Speicherung von String-Paaren (z. B. deutschen/englischen Texten zur Übersetzung von Begriffen). Ich gehe in diesem Abschnitt aber davon aus, dass wohl meistens Objekte gespeichert werden.

Erzeugen von Dictionary-Objekten

Ein Dictionary-Objekte erzeugen Sie folgendermaßen:

```
Dim dic
Set dic = Server.CreateObject("Scripting.Dictionary")
```

Anfügen von Elementen

Werte bzw. Objekte können Sie der Auflistung über die `Add`-Methode anfügen:

```
dic.Add Key, Item
```

Item ist der zu speichernde Wert bzw. das Objekt. Wenn Sie ein Objekt hinzufügen, wird nur ein Zeiger auf das Objekt gespeichert. *Key* ist ein Schlüssel, über den Sie das Element später ansprechen können. Als Schlüssel sind nicht nur Strings, sondern auch Zahlen (und wahrscheinlich noch andere Datentypen) möglich. Der Schlüssel muss innerhalb der gespeicherten Elemente eindeutig sein. Das neue Element wird an das Ende der Liste angefügt. Das folgende Beispiel erzeugt einige Personen-Objekte und fügt diese an ein Dictionary an:

```
<%
 ' Klasse zur Speicherung von Personendaten
 Class Person
   Public FirstName
   Public LastName
   Public Hobby
 End Class

 ' Erzeugen des Dictionary-Objekts
 Dim dicPersons
```

```
Set dicPersons = _
   Server.CreateObject("Scripting.Dictionary")

' Neue Person erzeugen und definieren
Set objPerson = New Person
objPerson.FirstName = "Zaphod"
objPerson.LastName = "Beeblebrox"
objPerson.Hobby = "Durchs Weltall reisen"
' und an die Auflistung anhängen;
' der Name wird als Schlüssel verwendet
dicPersons.Add objPerson.FirstName & _
  " " & objPerson.LastName, objPerson

' Weitere Person erzeugen und definieren
' Anmerkung: Das Erzeugen einer weiteren Person
' in objPerson entfernt zwar die Referenz auf
' die erste Person, da diese aber in der
' Auflistung referenziert ist, bleibt
' das Objekt erhalten
Set objPerson = New Person
objPerson.FirstName = "Ford"
objPerson.LastName = "Prefect"
objPerson.Hobby = "Durchs Weltall reisen"
dicPersons.Add objPerson.FirstName & _
   " " & objPerson.LastName, objPerson
%>
```

Automatisches Hinzufügen

Ein Feature der `Dictionary`-Klasse ist, dass diese ein neues Element automatisch anlegt, wenn der `Item`-Eigenschaft (die zum Zugriff auf einzelne Elemente verwendet wird) ein noch nicht vorhandener Schlüssel übergeben wird:

```
Dim dic
Set dic = Server.CreateObject("Scripting.Dictionary")
dic.Item("Auto") = "Ford Puma"
dic.Item("Haus") = "40qm Villa"
dic.Item("Katze") = "Wilde Hauskatze"
Response.Write dic.Item("Hund")
```

Dieses Feature müssen Sie natürlich im Auge behalten. Wenn Sie auf vorhandene Elemente zugreifen wollen und einen nicht vorhandenen Schlüssel übergeben, besitzt das `Dictionary`-Objekt ein neues Element mit einem leeren Variant-Wert (`Empty`). Der letzte `Response.Write`-Aufruf in dem obigen Beispiel erzeugt ein solches Element. Aber erfreulicherweise kennt ein `Dictionary`-Objekt eine `Exists`-Methode, mit der das Vorhandensein eines Elements mit einem bestimmten Schlüssel überprüft werden kann.

Löschen von Elementen

Für das Löschen von Elementen besitzt die `Dictionary`-Klasse die `Remove`-Methode:

```
dic.Remove Key
```

`Remove` löscht das Element, das den Schlüssel *Key* besitzt. Leider können Sie nur den Schlüssel angeben, nicht den Integer-Index des Elements (was z. B. bei der `Collection` geht):

```
' den Eintrag mit dem Schlüssel "Hund" löschen
dic.Remove "Hund"
```

Mit `RemoveAll` löschen Sie alle Elemente der Liste.

Der Zugriff auf die Elemente

Der Zugriff auf Elemente erfolgt über die `Item`-Eigenschaft unter Übergabe des Schlüssels:

```
MsgBox dic.Item("Hund")
```

Da `Item` die Standardeigenschaft ist, können Sie diese auch weglassen:

```
MsgBox dic("Hund")
```

Weil `Item` bei dem `Dictionary`-Objekt eine Eigenschaft ist, können Sie, im Gegensatz zur Collection, das Element an einem Schlüssel auch überschreiben.

```
dic.Item("Hund") = "Dog"
```

Sie können der `Item`*-Eigenschaft leider nur einen Schlüssel und keinen Integer-Index übergeben. Wenn Sie z. B. auf das erste Element zugreifen wollten und Sie übergeben die Zahl 1 als Schlüssel, legt das* `Dictionary`*-Objekt einfach ein neues Element mit dem Schlüssel 1 an, wenn noch kein Element mit diesem Schlüssel existiert.*

Überprüfen, ob Elemente existieren

Mit der `Exists`-Methode können Sie ermitteln, ob ein Eintrag mit einem bestimmten Schlüssel in der Liste gespeichert ist:

```
dic.Exists(Key)
```

Das Hobby von Zaphod Beeblebrox können Sie folgendermaßen abfragen:

```
If dicPersons.Exists("Zaphod Beeblebrox") Then
   Response.Write "Zaphod Beeblebrox hat das " & _
      "folgende Hobby: " & _
      dicPersons.Item("Zaphod Beeblebrox").Hobby
 Else
    Response.Write "Zaphod Beeblebrox existiert " & _
       "nicht in der Auflistung, aber das " & _
       "kann ja eigentlich gar nicht sein."
 End If
```

Sequenzielles Durchgehen

Wenn Sie ein `Dictionary`-Objekt sequenziell durchgehen wollen, können Sie zunächst keinen Integer-Index verwenden, da ein Dictionary-Objekt bei der `Item`-Eigenschaft einen Integer-Wert als Schlüssel interpretiert und einfach ein neues Element anlegt, wenn der Schlüssel nicht gefunden wird:

```
Dim dic
Set dic = Server.CreateObject("Scripting.Dictionary")
dic.Add "a", 1
dic.Add "b", 2
dic.Add "c", 3
Dim i
For i = 1 To dic.Count
```

```
   Response.Write dic.Item(i)
Next
```

Dieses Beispiel gibt nichts aus, da nicht die vorhandenen Elemente angesprochen, sondern einfach neue erzeugt werden.

Sie können ein `Dictionary`-Objekt direkt mit `For Each` durchgehen:

```
For Each objPerson In dicPersons.Items
   Response.Write objPerson.FirstName & "<br>"
Next
```

Beachten Sie, dass Sie die `Items`*-Eigenschaft angeben müssen. Lassen Sie diese Eigenschaft weg, gehen Sie nicht die Elemente durch, sondern die Schlüssel.*

Die `Items`-Methode gibt ein Array zurück, das alle Elemente des `Dictionary`-Objekts enthält. Die `Keys`-Methode leistet Ähnliches für die Schlüssel.

9 Interaktive Webseiten mit ASP

Internetanwendungen müssen nicht nur dynamisch sein, sondern oft auch interaktiv. Der Benutzer soll in der Lage sein, die Ausführung der Anwendung zu steuern, so wie es in Windowsanwendungen üblich ist. In Internetanwendungen, die mit ASP entwickelt werden, ist Interaktivität leider nicht ganz so einfach wie in normalen Windowsanwendungen. Dazu verwenden Sie Verweise (Links) zum Aufruf anderer ASP-Dokumente und HTML-Formulare, in der der Benutzer Daten eingeben oder auswählen kann. Ein Problem dabei ist immer, dass ASP-Dokumente grundsätzlich unabhängig voneinander ausgeführt werden. Die Übergabe von Daten an andere ASP-Seiten und das Speichern globaler Daten ist für interaktive Webseiten deshalb ebenfalls sehr wichtig.

In ASP.NET ist Interaktivität übrigens wesentlich einfacher zu programmieren, weil die meisten Programmiertechniken, die Sie in diesem Kapitel lernen, automatisch von ASP.NET verwendet werden. Zur Auswertung von Formularen erzeugen Sie z. B. nur eine einfache Ereignisprozedur und müssen sich nicht um das Senden und Empfangen der Daten kümmern. Da die zugrunde liegende Technik jedoch dieselbe ist wie bei ASP, ist das Wissen um diese Technik schon recht hilfreich. Sie sollten die einzelnen Themen dieses Kapitels also wenigstens lesen und kurz ausprobieren, um diese Technik zu erlernen. Wegen der Vereinfachung in ASP.NET gehe ich auch nicht mehr allzu detailliert auf die einzelnen Themen ein.

HiN/WEiS

Da ASP.NET einen großen Teil dieses Buchs einnimmt, musste ich einige ASP-Tipps und Tricks in einen Online-Artikel auslagern. Der Artikel »ASP-Tipps und Tricks« zeigt, wie Sie Eingaben server- und clientseitig validieren, wie Sie ein Formular an unterschiedliche ASP-Seiten senden, wie Sie E-Mails in ASP versenden und wie Sie Dateien zum Server uploaden.

9.1 Basiswissen

Etwas Basiswissen ist bei dem Verständnis der interaktiven ASP-Programmierung recht hilfreich. Ich beschreibe deswegen hier, wie interaktive ASP-Dokumente prinzipiell im Webserver ausgeführt werden.

Wenn ein Client ein HTML- oder ASP-Dokument abfragt, wird dieses, wie Sie ja bereits wissen, vom IIS eingelesen, ausgewertet und zum Client gesendet. Der Client ist für die Darstellung der HTML-Elemente zuständig. Soll das Programm interaktiv sein, d. h. auf Eingaben des Benutzers reagieren, können Sie auf der Clientseite zwar prinzipiell mit JavaScript arbeiten. Wenn in diese Interaktivität jedoch Ressourcen einbezogen sind, die nur auf dem Webserver-Rechner verfügbar sind (wie z. B. eine Datenbank), können Sie mit der clientseitigen Programmierung nichts bewirken. Es gibt dafür zwar auch Ausnahmen, wie z. B. der clientseitige Datenbankzugriff in VBScript mit Hilfe von ADO-Techniken (um genau zu sein: mit Hilfe von RDS, den Remote Data Services). Aber diese Lösungen erfordern meist den Internet Explorer als Browser und sind schwieriger zu programmieren. Eleganter ist immer eine serverseitige Lösung.

Das Problem dabei ist nur, dass der Webserver auf eine Eingabe des Benutzers reagieren muss. Und das sieht prinzipiell so aus, dass diese Eingabe von einem ASP-Dokument ausgewertet wird. Dieses Dokument können Sie über einen Link oder über ein Formular aufrufen. Einem Link können Sie dazu beliebige Argumente übergeben, die Sie in der aufgerufenen ASP-Seite auswerten können. Ein Formular besitzt dagegen den Vorteil, dass der Benutzer die übertragenen Daten selbst eingeben kann. Auf einer Suchseite kann er so Suchkriterien für eine Datenbankabfrage eingeben.

Links mit URL-Argumenten sind sehr einfach: Die Argumente werden einfach an die URL angehängt. Die Übertragung von Formulardaten dagegen war für mich anfangs eher »magisch«, ist aber eigentlich ganz einfach.

Wenn ein HTML-Dokument ein Formular enthält und der Anwender sendet dieses über den Submit-Schalter (oder über eine andere Technik) ab, werden die Daten ja nach der Methode des Formulars unterschiedlich an den Server übertragen. Ist im `method`-Attribut des

form-Tag "Post" eingetragen, werden die Eingaben im HTTP-Body der Anforderung übertragen, bei der Methode "Get" verwendet der Browser URL-Argumente für die Übertragung. URL-Argumente sind eigentlich für solche Eingaben nicht geeignet, da die URL häufig auch in Formularen für eigene Steuerungszwecke verwendet wird. Ich gehe also in den folgenden Beispielen davon aus, dass Sie Ihre Formulare mit der Post-Methode versenden.

Das folgende Beispiel zeigt ein einfaches Formular und die HTTP-Anforderung, die durch die Bestätigung des Formulars erzeugt wird:

```
<form method="post" action="auswerten.asp">
 <table>
 <tr>
  <td>Vorname:</td>
  <td><input type="Text" name="Vorname"
   size="35"></td></tr>
 <tr>
  <td>E-Mail:</td>
  <td><input type="Text" name="EMail" size="35"></td>
 </tr>
 <tr>
  <td>Hobby:</td>
  <td><input type="Text" name="Hobby" size="35"></td>
 </tr>
 </table>
 <input type="submit" value="Absenden">
</form>
```

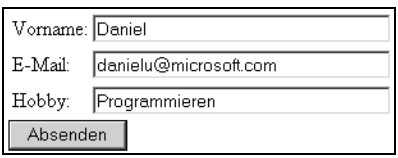


Bild 9.1: Das HTML-Formular für das Beispiel

Die durch das Absenden des Formulars erzeugte HTTP-Anforderung sieht in etwa so aus:

```
POST /Shop/Login.asp HTTP/1.1.
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applica-
tion/msword, */*.
Referer: http://Trillian/Shop/Login.htm.
Accept-Language: de.
Content-Type: application/x-www-form-urlencoded.
Accept-Encoding: gzip, deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0).
Host: Trillian.
Content-Length: 64.
Connection: Keep-Alive.
Cache-Control: no-cache.
.
Vorname=Daniel&EMail=danielu@microsoft.com&Hobby=Programmieren
```

Sie sehen, dass die Formulardaten im Body der Anforderung an den Server übertragen werden. Einzelne Datenfelder werden durch ein Ampersand getrennt. In ASP können Sie die so empfangenen Daten sehr einfach über `Request.Form` auswerten.

So erhält der Client also die eingegebenen Daten. Ist doch ganz einfach, oder?

9.2 URL-Argumente auswerten

Eine beliebte Technik, aufgerufenen ASP-Dokumenten Daten zu übergeben, ist, der URL Argumente mitzugeben. So können Sie in einem ASP- oder HTML-Dokument Links unterbringen, die andere ASP-Dateien aufrufen und diesen Daten in der URL übergeben. Beispielsweise könnten Sie in einem Online-Shop Verweise unterbringen, die eine ASP-Datei zur Anzeige von Produkten jeweils mit den einzelnen Produkt-Kategorien aufruft, damit nur die Produkte einer Kategorie angezeigt werden.

URLs mit Argumenten werden aber nicht nur für Verweise, sondern auch für das gesteuerte Bestätigen von Formularen (im `Action`-Argument des `form`-Tags) verwendet. Da innerhalb von ASP-Dokumenten natürlich auch URLs dynamisch aufgebaut werden können, ist diese Technik sehr flexibel.

Eine URL mit einem Argument baut sich folgendermaßen auf:

Ziel?*Argumentname=Wert*

Das Fragezeichen trennt die Zielangabe vom Argument. Mehrere Argumente trennen Sie mit einem Ampersand (&):

Ziel?*Argumentname1=Wert&Argumentname2=Wert*

Beachten Sie, dass die URL bestimmten HTTP-Regeln entsprechen muss, damit alle Browser damit zurechtkommen. Leerzeichen müssen als + dargestellt werden, Sonderzeichen mit ihrem hexadezimal dargestellten ANSI-Wert. Wenn Sie innerhalb von ASP eine URL dynamisch aufbauen, sollten Sie dazu immer die `URLEncode`*-Methode des* `Server`*-Objekts verwenden.*

Ein Link mit einer URL, der ein Dokument Products.asp aufruft und dabei eine Kategorienummer und Minimal- und Maximalpreis übergibt, sieht z. B. so aus:

```
<a
 href="Products.asp?CategoryId=7&MinPrice=0&MaxPrice=25">
 Naturprodukte bis 25 Euro
</a>
```

Der Name der Argumente kann natürlich frei vergeben werden.

In der aufgerufenen ASP-Seite werten Sie URL-Argumente über die `QueryString`-Auflistung des `Request`-Objekts aus:

```
Response.Write "Sie forderten Produkte der " & _
   "Kategorie " & Request.QueryString("CategoryId") & _
   " im Preis zwischen " & _
   Request.QueryString("MinPrice") & " und " & _
   Request.QueryString("MaxPrice") & " Euro an."
```

Auf leere bzw. ausgelassene Argumente reagieren

In vielen Fällen werden Sie URL-Argumente auch einmal weglassen. Die Products.asp-Datei des Beispiels könnte auch ohne Minimal- und Maximalpreis aufgerufen werden:

```
<a href="Products.asp?CategoryId=1">
 Getränke
</a>
```

Normalerweise müssen Sie auf ausgelassene Argumente reagieren. Im Prinzip ist das ganz einfach: Die `QueryString`-Auflistung ist eine normale Auflistung. Wenn Sie auf ein Element zugreifen wollen, das noch nicht existiert, legt die Auflistung automatisch ein neues Element an. Da dieses Element uninitialisiert ist, speichert es (in ASP, nicht in ASP.NET!) den Wert `Empty`. Sie können also mit der VBScript-Funktion `IsEmpty` abfragen, ob ein Element, also ein URL-Argument, leer ist:

```
If IsEmpty(Request.QueryString("MinPrice")) Then
   Response.Write "Sie forderten Produkte der " & _
      "Kategorie " & Request.QueryString("CategoryId") & _
      " an."
Else
   Response.Write "Sie forderten Produkte der " & _
      "Kategorie " & Request.QueryString("CategoryId") & _
      " im Preis zwischen " & _
      Request.QueryString.Item("MinPrice") & " und " & _
      Request.QueryString.Item("MaxPrice") & " Euro an."
End If
```

Eine andere Möglichkeit, die auch in ASP.NET funktioniert, ist, das Argument mit einem Leerstring zu vergleichen. VBScript konvertiert `Empty` bei Stringvergleichen implizit in einen Leerstring um, so dass dieser Vergleich funktioniert. Das folgende Beispiel kopiert den Wert des URL-Arguments zuvor in eine Variable:

```
Dim strMinPrice
strMinPrice = Request.QueryString("MinPrice")
If strMinPrice = "" Then
   ...
```

Gleichnamige Argumente

Manchmal macht es vielleicht Sinn, Argumente gleich zu benennen. Einer URL können Sie z. B. eine Liste von Namen übergeben, die dynamisch zusammengestellt wurde (weswegen Sie dann nicht wissen,

wie viele Namen übergeben wurden). Benennen Sie die einzelnen Argumente dann einfach gleich:

```
<a href="Namensliste.asp?Name=Zaphod&Name=Trillian&
Name=Ford&Name=Arthur">
 Namensliste
</a>
```

`Request.QueryString` gibt in diesem Fall eine Auflistung zurück. Damit können Sie in ASP (nicht in ASP.NET!) über einen Integer-Index auf die Elemente zugreifen:

```
Dim i
For i = 1 To Request.QueryString("Name").Count
    Response.Write Request.QueryString("Name")(i) & "<br>"
Next
```

Alternativ (und einfacher) ist die Verwendung der `For-Each`-Schleife:

```
Dim strName
For Each strName in Request.QueryString("Name")
   Response.Write strName & "<br>"
Next
```

In ASP.NET können Sie diese Syntax nicht verwenden, weil `QueryString` *dort im Fall von Array-Argumenten einen einfachen String zurückgibt. Dieser enthält die einzelnen Argumente, getrennt durch Kommata. Ich konnte nicht herausfinden, ob und wie ASP.NET das Durchgehen der einzelnen Argumente erlaubt.*

Als Behelf habe ich in ASP.NET einfach die VB-Funktion `Split` verwendet. Das funktioniert aber nur, solange die übergebenen Argumente selbst kein Komma enthalten:

```
Dim astrArgs() As String
astrArgs = Split(Request.QueryString.Item("Name"), ",")
Dim strName As String
For Each strName In astrArgs
    Response.Write(strName & "<br>")
Next
```

9.3 Arbeiten mit HTML-Formularen

9.3.1 Eingaben auswerten

Die Auswertung von Eingaben, die ein Benutzer in einem HTML-Formular vorgenommen hat, ist eine sehr wichtige Aufgabe bei der ASP-Programmierung. Besonders bei Datenbankanwendungen werden Formulare sehr intensiv genutzt. Über ein Formular kann ein Benutzer z. B. Datensätze abfragen (siehe Kapitel 10), über ein anderes Formular neue Datensätze anfügen.

HTML-Formulare wurden ja bereits in Kapitel 4 behandelt. Hier geht es allein um die Auswertung. Und die ist eigentlich ganz einfach.

Ähnlich wie bei URL-Argumenten erhalten Sie Zugriff auf Formulareingaben über die `Form`-Auflistung des `Request`-Objekts. Als Schlüssel geben Sie den Namen des auszuwertenden Steuerelements an. Ein einfaches Formular wie in Abbildung 9.1 auf Seite 231 werten Sie so aus:

```
Dim strVorname, strEMail, strHobby
strVorname = Request.Form("Vorname")
strEMail = Request.Form("EMail")
strHobby = Request.Form("Hobby")

If strVorname = "" Then strVorname = "Nobody"
If strEMail = "" Then strEMail = "nicht verfügbar"
If strHobby = "" Then strHobby = "scheinbar Schlafen"

Response.Write "Hallo " & strVorname & _
  ". Deine E-Mail-Adresse ist " & strEMail & _
  ". Dein Hobby ist " & strHobby & "."
```

Was Sie an diesem Beispiel schon erkennen können, ist die Tatsache, dass die Überprüfung von Eingaben auf der Serverseite leider nicht allzu einfach ist. Viele Programmierer überprüfen Eingaben deshalb bereits auf der Clientseite (mit JavaScript im Browser). Wenn Sie mit ASP.NET programmieren, sollten Sie zur Validierung besser die speziellen Validierungs-Steuerelemente verwenden (Kapitel 13).

9.3.2 Auswerten der verschiedenen Steuerelemente

Einige Steuerelemente wie Textfelder sind recht einfach auszuwerten. Andere, wie Mehrfachauswahl-Listen, erfordern ein wenig mehr Aufwand. Ich zeige deshalb kurz, wie Sie die verschiedenen Steuerelemente auswerten. Grundsätzlich sollten Sie beachten, dass der Browser beim Senden der Daten in der Regel das `name`-Attribut der Steuerelemente verwendet, nicht das `id`-Attribut (das ja hauptsächlich in DHTML Verwendung findet).

Leere Eingaben können Sie einfach (wie beim Auswerten von URL-Argumenten) über einen Vergleich mit einem Leerstring ermitteln.

Textfelder

Bei der Auswertung von Textfeldern in ASP erhalten Sie den im Textfeld gespeicherten Wert. Die Auswertung ist also ganz einfach:

```
Dim strName
strName = Request.Form("Name")
If strName > "" Then
   Response.Write strName & "<br>"
Else
   Response.Write "Sie haben keinen Namen angegeben.<br>"
End If
```

Bei Mehrzeilen-Textfeldern gibt `Request.Form` alle eingegebenen Zeilenumbrüche als solche zurück. Wenn Sie diese Eingaben in HTML ausgeben wollen, sollten Sie die Zeilenumbrüche durch den `br`-Tag ersetzen. Ein Mehrzeilen-Adressfeld werten Sie so aus:

```
Dim strAdresse
strAdresse = Request.Form("adresse")
Response.Write Replace(strAdresse, vbCrLf, "<br>")
```

Achten Sie bei `textarea`*-Elementen darauf, dass diese in HTML keine ungewollten Leerzeichen enthalten.*

```
<textarea name="adresse" id="adresse">
</textarea>
```

Wenn Sie ein `textarea`-Element z. B. folgendermaßen anlegen, enthält das Steuerelement bereits Leerzeichen, nämlich die, die vor dem abschließenden Tag stehen. Legen Sie `textarea`-Elemente also immer entweder linksbündig im Quelltext an, oder hängen Sie den abschließenden Tag idealerweise direkt an:

```
<textarea name="adresse" id="adresse"></textarea>
```

Listenfelder

Einfachauswahl-Listenfelder geben einfach den ausgewählten Wert zurück. Auch wenn der Anwender gar nichts ausgewählt hat, erhalten Sie einen Wert, nämlich den des ersten Eintrags. Um dem Anwender die Möglichkeit zu geben, auch keinen Eintrag auszuwählen, müssen Sie einen leeren Eintrag in der Liste erzeugen:

```
<select name="fahrzeug" id="fahrzeug">
  <option></option>
  <option>Fahrrad</option>
  <option>Motorrad</option>
  <option>Auto</option>
  <option>Bus</option>
</select>
```

Wählt der Anwender den leeren Eintrag aus, erhalten Sie einen Leerstring. Die Auswertung sieht dann so aus wie bei Textfeldern:

```
Dim strFahrzeug
strFahrzeug = Request.Form("fahrzeug")
If strFahrzeug > "" Then
   Response.Write strFahrzeug & "<br>"
Else
   Response.Write "Sie haben kein Fahrzeug angegeben.<br>"
End If
```

Mehrfachauswahl-Listenfelder geben die ausgewählten Einträge in einer kommabegrenzten Liste zurück, wenn Sie die `Form`-Auflistung direkt abfragen. Über eine direkte Abfrage können Sie ermitteln, ob überhaupt etwas ausgewählt wurde. Die Rückgabe ist tatsächlich aber eine Auflistung, die Sie wie gewohnt über einen Integer-Index oder mit `For-Each` durchgehen können. Das folgende Listenfeld

```
<select name="hobbys" id="hobbys" multiple="multiple">
 <option>Carven, Snowboarden</option>
 <option>Inlinern</option>
 <option>Biken</option>
</select>
```

werten Sie z. B. so aus:

```
Dim strHobby
If Request.Form("hobbys") > "" Then
   For Each strHobby In Request.Form("hobbys")
      Response.Write strHobby & "<br>"
   Next
Else
   Response.Write "Sie haben kein Hobby angegeben.<br>"
End If
```

Normalerweise erhalten Sie bei Listenfeldern den Inhalt der `option`*-Tags zurück. Wenn Sie für diese Tags aber das Attribut* `value` *definieren, erhalten Sie den dort angegebenen Wert. Das macht die Auswertung dann unabhängig vom ausgegebenen Text.*

```
<select name="hobbys" id="hobbys" multiple="multiple">
 <option value="SnowFun">Carven, Snowboarden</option>
 <option value="StreetFun">Inlinern</option>
 <option value="HillFun">Biken</option>
</select>
```

Optionbuttons

Wenn Sie Optionbuttons einzeln anlegen (also nicht über den Namen gruppieren), erhalten Sie bei der Auswertung den Wert »on« zurück, wenn der Optionbutton eingeschaltet ist. Normalerweise legen Sie aber Gruppen von Optionbuttons an, deren einzelne Elemente denselben Namen besitzen. Um diese auswerten zu können, müssen Sie in einzelnen `input`-Tags das Attribut `value` definieren. Ist der Button ausgewählt, erhalten Sie dann den Wert dieses Attributs zurück. Die Lieblings-Musikrichtung des Anwenders können Sie z. B. so eingeben lassen:

```
<input type="radio" name="musik" value="Alternative"
 checked="checked">Alternative<br>
<input type="radio" name="musik"
 value="Electronic">Electronic<br>
<input type="radio" name="musik"
 value="HipHop">HipHop<br>
<input type="radio" name="musik" value="Jazz">Jazz<br>
<input type="radio" name="musik" value="Blues">Blues<br>
<input type="radio" name="musik" value="Heimatmusik"
 disabled="disabled">Heimatmusik<br>
```

Aus irgendeinem Grund ist die Option »Heimatmusik« in diesem Beispiel deaktiviert. Ich habe keine Ahnung, warum. Ehrlich.

Sie sollten immer eine Option voreinstellen, wie im Beispiel die Option »Alternative«. Dann können Sie sicher sein, dass auch eine Option ausgewählt ist.

Die Auswertung ist dann ganz einfach:

```
Response.Write Request.Form("musik") & "<br>"
```


Checkboxen

Bei der Auswertung von Checkboxen müssen Sie häufig die einzelnen Eingaben auswerten. Dazu stehen Ihnen grundsätzlich zwei Möglichkeiten zur Verfügung: Entweder vergeben Sie allen Checkboxen einen individuellen Namen und werten diese separat aus oder Sie vergeben denselben Namen und werten die Eingaben gemeinsam aus. Besitzen mehrere Checkboxen denselben Namen, gibt die `Form`-Auflistung eine Auflistung zurück, die die Werte der eingeschalteten Checkboxen enthält. Der jeweils zurückgegebene Wert ist »on«, wenn die Checkbox kein `value`-Attribut besitzt, ansonsten der Wert des `value`-Attributs.

Individuelle Namen erleichtern das gezielte, individuelle Auswerten:

```
<!-- Checkboxen mit individuellem Namen -->
<input type="checkbox" id="windows"
 name="windows">Windows</input><br>
<input type="checkbox" id="unix"
 name="unix">Unix</input><br>
```

```
<input type="checkbox" id="linux"
 name="linux">Linux</input><br>
...

' Gezieltes Auswerten von Checkboxen mit
' individuellem Namen
If Request.Form("windows") = "on" Then
   Response.Write "Windows<br>"
End If
If Request.Form("unix") = "on" Then
  Response.Write "Unix<br>"
End If
If Request.Form("linux") = "on" Then
   Response.Write "Linux<br>"
End If
```

Ein gemeinsamer Name erleichtert das gemeinsame Auswerten.

```
<!-- Checkboxen mit identischem Namen -->
<input type="checkbox" id="betriebssystem"
 name="betriebssystem" value="Windows">Windows</input><br>
<input type="checkbox" id="betriebssystem"
 name="betriebssystem" value="Unix">Unix</input><br>
<input type="checkbox" id="betriebssystem"
 name="betriebssystem" value="Linux">Linux</input><br>
```

Sinnvoll ist dann das Durchgehen der ausgewählten Optionen mit `For-Each`:

```
' Auswerten von Checkboxen mit gemeinsamen Namen
Dim strBetriebssystem
If Request.Form("betriebssystem") > "" Then
   For Each strBetriebssystem In
      Request.Form("betriebssystem")
      Response.Write strBetriebssystem & "<br>"
   Next
Else
   Response.Write "Sie haben kein " & _
      "Betriebssystem angegeben.<br>"
End If
```

9.3.3 Versteckte Daten in hidden-Elementen übertragen

ASP-Dokumente, die von einem Formular aus aufgerufen werden, müssen häufig Informationen erhalten, die der Anwender nicht sehen soll. Ein Bestellformular in einem Online-Shop muss z. B. die ID des Kunden an das ASP-Dokument zur Speicherung der Bestellung übergeben. Prinzipiell können Sie dazu Werte verwenden, die im `Session`-Objekt gespeichert sind. Wenn der Browser des Benutzers jedoch keine Cookies unterstützt, fällt das `Session`-Objekt für die Übergabe von Werten schon einmal weg. Die Übergabe von Daten in URL-Argumenten ist für diesen Fall auch nicht optimal, weil der Benutzer diese dann sehen kann.

Die Lösung dieses Problems sind versteckte HTML-Elemente, die auf dem Formular angelegt werden. Ein solches Element wird ja, wie Sie bereits wissen, ähnlich einem Textfeld angelegt. Im `value`-Attribut speichern Sie den zu übergebenden Wert:

```
<form method="post" action="Bestellung_Speichern.asp">
 Artikel-Nr:<br>
  <input type="text" name="ArtikelNr" id="ArtikelNr"><br>
 Anzahl:<br>
  <input type="text" name="Anzahl" id="Anzahl"
    value="1"><br>
 <input type="hidden" name="UserId" id="UserId"
   value="<% = userId%>">
</form>
```

Das Beispiel schreibt den Wert der Variablen *userId* in das versteckte Feld.

Bei der Auswertung des Formulars werten Sie die so erzeugten Daten aus wie bei einem normalen Textfeld.

Versteckte Felder können Sie für sehr viele Tricks nutzen. Immer dann, wenn Sie Daten versteckt von einem Formular aus an eine ASP-Seite übergeben wollen, nutzen Sie diese Felder. Damit können Sie einen bestimmten Status auch über mehrere ASP-Seiten verwalten, ohne dazu das `Session`-Objekt zu verwenden. Ein beliebter »Trick« dabei ist, empfangene versteckte Daten für den »Weitertransport« wieder in versteckte Elemente zu schreiben. Die ASP-Seite zum

Speichern der Bestellung könnte die empfangene User-Id z. B. selbst wieder speichern, damit diese weitergegeben werden kann:

```
<input type="hidden" name="UserId" id="UserId"
  value="<% = Request.Form("UserId")%>">
```

Leider können Sie diese Felder nur in Formularen übergeben. ASP-Dokumente, die keine Formulare enthalten, können solche Daten also nicht weitergeben.

Beachten Sie, dass Daten in `hidden`*-Elementen nicht wirklich versteckt sind. Der Benutzer kann einfach den Quelltext der Seite im Browser anzeigen lassen und sieht dann natürlich auch alle versteckten Elemente.*

9.4 Das Application-Objekt

Die hauptsächliche Aufgabe des `Application`-Objekts ist die Speicherung von Daten, die von allen Clients verwendet werden können.

9.4.1 Globale Daten speichern

Zur Speicherung globaler Daten besitzt das `Application`-Objekt eine Auflistung `Contents`, der Sie beliebige Daten hinzufügen können:

```
Application.Contents.Item("Schlüssel") = Wert
```

In vielen Beispielen wird auf die Angabe der `Item`-Methode verzichtet:

```
Application("Schlüssel") = Wert
```

In beliebigen ASP-Seiten der Anwendung können Sie diese Daten dann auslesen:

```
Response.Write Application.Contents.Item("Schlüssel")
```

Da `Item` die Defaulteigenschaft von `Contents` und `Contents` die Defaulteigenschaft des `Application`-Objekts ist, können Sie auch eine Kurzform verwenden:

```
Application("Schlüssel") = Wert
```

In ASP.NET können Sie die verkürzte Schreibweise nicht verwenden. Dort können Sie über `Contents.Item` *auf die Auflistung zugreifen, aber auch einfach die* `Item`*-Methode direkt verwenden (die in ASP.NET dem* `Application`*-Objekt direkt untergeordnet ist).*

`Contents` *ist ein* `Dictionary`*-Objekt. Objekte dieser Klasse besitzen das Feature, dass die* `Item`*-Eigenschaft ein Element automatisch anlegt, falls die Auflistung noch kein Element mit diesem Schlüssel besitzt. Deswegen funktioniert die Speicherung, ohne dass ein Element explizit angefügt wird.*

Sie können im `Application`-Objekt auch Objekte speichern, dann müssen Sie allerdings mit `Set` zuweisen:

```
Set Application.Value("Schlüssel") = Objekt
```

Um zu verhindern, dass zwei ASP-Programme quasi gleichzeitig in das `Application`-Objekt schreiben (ASP-Seiten werden in Threads ausgeführt!), können Sie den Zugriff mit der `Lock`-Methode sperren und nach dem Setzen mit `Unlock` wieder freigeben:

```
Application.Lock
Application.Contents.Item("LastAccess") = Now
Application.Contents.Item("HitCounter") = _
   Application.Contents.Item("HitCounter") + 1
Application.Unlock
```

Ohne ein Sperren könnte es bei vielen gleichzeitigen Benutzern vorkommen, dass zwei ASP-Seiten gleichzeitig in Elemente der Auflistung schreiben und in besonderen Fällen ungültige Daten verursachen würden. Da ich mir nicht vorstellen kann, dass ein Thread mitten im Schreiben eines Wertes unterbrochen wird, wird dies aber ganz von Ihrer Programmierung abhängen. Wenn Sie z. B. in ein `Application`-Element einen Wert schreiben, den Sie weiter unten im Programm noch einmal lesen, kann es sein, dass der Wert zwischenzeitlich von einer anderen ASP-Seite geändert wurde. Setzen Sie also immer `Lock` und `Unlock` ein, um sicher zu gehen. Wenn der Zugriff gesperrt ist, müssen andere ASP-Threads so lange warten, bis der Zugriff wieder freigegeben wurde. Falls Sie einmal vergessen, die `Un-`

lock-Methode nach dem Schreiben aufzurufen, oder falls das Programm mitten im Schreiben abstürzt, ruft der IIS diese Methode implizit auf, wenn die Abarbeitung der Seite beendet wurde, oder wenn das Script-Timeout abgelaufen ist.

9.4.2 Löschen gespeicherter Daten

Mit der Remove-Methode der Contents-Auflistung können Sie einzelne gespeichert Daten, mit der RemoveAll-Methode alle gespeicherten Daten löschen. Das Löschen von Daten macht dann Sinn, wenn eine ASP-Anwendung sehr viele Daten im Application-Objekt speichert, die aber nur zeitweise benötigt werden. Um Ressourcen zu sparen, sollten Sie diese Daten regelmäßig löschen.

Über die Remove-Methode können Sie einzelne Elemente über deren Schlüssel oder über einen Integer-Index löschen:

```
Application.Contents.Remove(Key | Index)
```

Wenn Sie über einen Integer-Index löschen, geben Sie als Argument die Position des Elements in der Auflistung an. Normalerweise dürfte das aber eher selten der Fall sein, meistens wird ein Element wohl über den Schlüssel des Elements gelöscht.

Die RemoveAll-Methode löscht alle Elemente der Auflistung:

```
Application.Contents.RemoveAll
```

9.4.3 Auswerten der Ereignisse OnStart und OnEnd

Das Application-Objekt besitzt die beiden Ereignisse OnStart und OnEnd, über die Sie den Start und das Beenden der Applikation auswerten können. Eine Applikation startet dann, wenn der erste Benutzer irgendeine Datei von dem Webordner abruft, nachdem die Applikation neu initialisiert wurde. Neu initialisiert wird eine Applikation immer dann, nachdem der Webordner neu erzeugt wurde, wenn der Webserver neu gestartet wurde oder nachdem die Datei GLOBAL.ASA geändert wurde. Beendet wird eine Applikation immer dann, wenn der Webserver heruntergefahren wird oder nachdem die Datei GLOBAL.ASA verändert wurde.

Im `OnStart`-Ereignis können Sie also die gesamte Applikation initialisieren. Viele Programmierer schreiben hier Werte in die `Application`-Auflistung, die dann im Programm als Voreinstellung für irgendwelche Aktionen verwendet werden. Denkbar wäre auch noch die Erzeugung von Objekten, die Sie im Programm öfter benötigen, und die Speicherung derselben in der `Application`-Auflistung. Sinnvoll ist dieses Vorgehen jedoch nur in Intranets, wenn nur relativ wenige Benutzer gleichzeitig auf den Webordner zugreifen. Da über diese Technik nur *ein* einziges Objekt für *alle* Benutzer erzeugt wird, wird dieses Vorgehen im Internet zu einem enormen Performance-Nachteil führen, nämlich dann, wenn vielleicht mehrere Hundert Benutzer gleichzeitig eine ASP-Seite abrufen, die ein solches Objekt verwendet. Im `OnEnd`-Ereignis können Sie Ressourcen, die Sie im `OnStart`-Ereignis geöffnet haben, wieder freigeben. Dieses Ereignis wird eigentlich fast nie benötigt, da ein sauberes ASP-Programm möglichst keine Ressourcen über die gesamte Lebensdauer der Applikation geöffnet hält.

Jetzt stellt sich nur noch die Frage, wo Sie die Ereignisprozeduren für diese Ereignisse deklarieren. Die Antwort: 42. Nein, das war nur ein Scherz. Diese Ereignisprozeduren müssen Sie in einer Datei mit dem Namen GLOBAL.ASA deklarieren. Diese spezielle Datei ist für die Ereignisprozeduren des `Session`- und des `Application`-Objekts und für spezielle Objektdeklarationen vorgesehen. Damit der IIS diese Datei findet, muss sie im Stammverzeichnis des Webordners gespeichert sein.

9.5 Sitzungen

Wie Sie ja bereits wissen, erzeugt der IIS bzw. das .NET-Framework (bei ASP.NET) für jede HTTP-Anforderung, für die noch keine Sitzung besteht, eine neue Sitzung, sofern in der Konfiguration des Webordners Sitzungen ermöglicht sind.

Bei ASP ermöglichen Sie Sitzungen dadurch, dass der Webordner als Anwendung konfiguriert ist und dass die Option SITZUNGSSTATUS AKTIVIEREN für diese Anwendung aktiviert ist. Bei ASP.NET verwalten Sie die Sitzungs-Konfiguration in der Datei WEB.CONFIG, die im Stam-

mordner des Webordners gespeichert sein muss. Auf die ASP.NET-Konfiguration gehe ich in Kapitel 11 ein.

Sitzungen ermöglichen das globale Speichern von Daten so, dass diese innerhalb aller ASP-Seiten einer Anwendung verwendet werden können. Die Daten sind nur innerhalb der jeweiligen Sitzung verfügbar. Andere Sitzungen haben keinen Zugriff auf die so gespeicherten Daten. Sitzungs-Daten bleiben so lange bestehen, bis die Sitzung beendet wird. Eine Sitzung wird erst dann beendet, wenn der Benutzer so lange kein weiteres Dokument vom Webordner abgerufen hat, wie in der Konfigurations-Einstellung SITZUNGS-TIMEOUT der Anwendung angegeben ist. Diese Einstellung finden Sie bei ASP in der Konfiguration der Webanwendung in der IIS-Administration und bei ASP.NET in der Datei WEB.CONFIG. Für ASP-Dokumente, bei denen Sie erwarten, dass der Benutzer diese lange geöffnet hat (z. B. um umfangreiche Daten zu bearbeiten), können Sie den Timeout über `Session.Timeout` auch auf andere Werte einstellen (siehe Seite 253).

Speichern von Daten

Das Speichern von Daten im `Session`-Objekt ist sehr einfach. Fügen Sie die zu speichernden Daten einfach über die `Item`-Methode der `Contents`-Auflistung an:

```
Session.Contents.Item("StartTime") = Time
```

In vielen Beispielen wird auf die Angabe der `Item`-Methode verzichtet:

```
Session("StartTime") = Time
```

Das funktioniert in VBScript, weil `Item` die Default-Methode der `Contents`-Auflistung und diese die Default-Eigenschaft des `Session`-Objekts ist.

In ASP.NET können Sie die verkürzte Schreibweise nicht verwenden. Dort können Sie über `Contents.Item` auf die Auflistung zugreifen, aber auch einfach die `Item`-Methode direkt verwenden (die in ASP.NET dem `Session`-Objekt direkt untergeordnet ist):

```
Session.Item("StartTime") = Time ' ASP.NET
```

Das Speichern von Daten und Objekten im `Session`*-Objekt funktioniert unter ASP nur dann, wenn der Browser Cookies unterstützt (siehe Seite 250).*

Viele ASP-Programme verwalten über das `Session`-Objekt auch Objekte. Der Unterschied zum normalen Speichern ist unter ASP (nicht unter ASP.NET), dass Sie Objekte immer mit `Set` setzen müssen.

Das folgende Beispiel speichert ein `Dictionary`-Objekt in der `Session`-Auflistung:

```
Dim dicPersonen
Set dicPersonen = _
   Server.CreateObject("Scripting.Dictionary")
dicPersonen.Add "Donald", "Donald Duck"
dicPersonen.Add "Daisy", "Daisy Duck"
dicPersonen.Add "Dagobert", "Dagobert Duck"
Set Session("Personen") = dicPersonen
```

Beachten Sie, dass `Set` unter ASP.NET wegfällt. Beim Lesen solcher Objekte verwenden Sie die übliche Objektsyntax:

```
Dim strPerson
For Each strPerson In Session("Personen").Items
   Response.Write strPerson & "<br>"
Next
```

Beachten Sie, dass Sie nur dann auf Objekte zugreifen können, wenn diese auch erzeugt wurden. Existiert ein Objekt nicht, erhalten Sie einen Laufzeitfehler. Sie sollten deshalb immer überprüfen, ob das Objekt auch existiert. Das können Sie mit der `IsObject`*-Funktion erledigen:*

```
If IsObject(Session("Personen")) Then
   For Each strPerson In Session("Personen").Items
      Response.Write strPerson & "<br>"
   Next
Else
   Response.Write "Das Personen-Objekt " & _
      "existiert nicht.<br>"
End If
```

Mit Objekten, deren Klasse innerhalb des ASP-Dokuments deklariert ist, funktioniert das Speichern eigenartigerweise nicht. Das `Session`*-Objekt scheint das Objekt am Ende der ASP-Seite wieder zu zerstören. Beim Zugriff innerhalb anderer ASP-Dokumente auf das vermeintlich gespeicherte Objekt erzeugt VBScript einen Laufzeitfehler, der besagt, dass das Objekt die verwendete Eigenschaft oder Methode nicht besitzt. Das deutet darauf hin, dass das Objekt gar nicht mehr existiert. Für dieses Problem habe ich keine Lösung gefunden.*

Die Lebensdauer von Sitzungs-Objekten

Sitzungs-Objekte leben nur so lange, wie die Sitzung lebt. Das kann zum Problem werden, wenn Sie Objekte nicht im Startereignis der Sitzung, sondern in einer ASP-Seite (vielleicht der Startseite) der Anwendung erzeugen. Wartet der Anwender zu lange, so dass das Timeout abläuft, und greift dann wieder auf ein ASP-Dokument zu, sind die Sitzungsobjekte nicht mehr verfügbar.

Sie sollten Objekte, die Sie in einer Sitzung benötigen, also immer im Startereignis der Sitzung erzeugen.

Wenn Sie die Objekte explizit zerstören wollen, um Ressourcen zu sparen, können Sie das Sitzungs-Element auf `Nothing` setzen:

```
Set Session("Personen") = Nothing
```

Sitzung initialisieren

Wenn Sie eine Sitzung initialisieren wollen, verwenden Sie das Startereignis der Sitzung (`OnStart` in ASP, `Start` in ASP.NET) in den Dateien GLOBAL.ASA (ASP) bzw. GLOBAL.ASAX (ASP.NET). Im Ende-Ereignis können Sie Ressourcen, die Sie im Startereignis reserviert haben, wieder freigeben. Das explizite Freigeben von Objekten im Ende-Ereignis, das man immer wieder in ASP-Quellcodes sieht, ist übrigens nicht notwendig. Unter ASP und ASP.NET werden Objekte immer automatisch zerstört, sobald diese nicht länger benötigt werden.

Das folgende Beispiel erzeugt im Startereignis ein `Dictionary`-Objekt für die Speicherung von Bestellungen und speichert die Startzeit :

```
<script language="VBScript" runat="Server">
 Sub Session_OnStart()
  Set Session("Bestellungen") = _
   Server.CreateObject("Scripting.Dictionary")
  Session("Startzeit") = Now
 End Sub
</script>
```

Sitzungen und Cookies

Ein Cookie ist eine Speichereinheit, die über den Browser auf dem Rechner des Benutzers abgelegt wird (im Fall von Sitzungs-Cookies temporär im RAM). Cookies behandle ich noch ab Seite 254 näher. ASP legt die Id der Sitzung in einem Cookie ab. Jeder Browser sendet alle Cookies einer Webanwendung bei jeder Anforderung automatisch mit. ASP kann die Sitzungs-Id also aus den Cookies auslesen und die Sitzung damit identifizieren. In ASP.NET können Sie Sitzungen aber auch ohne Cookies verwalten. Setzen Sie dazu das Attribut `cookieless` im Element `sessionstate` der Konfigurationsdatei (WEB.CONFIG, siehe Kapitel 11) auf `True`. ASP.NET sorgt dann automatisch dafür, dass die Sitzungs-ID in jeder URL in deren Mitte mitgesendet und dort auch wieder ausgelesen wird. Damit können Sie Sitzungen auch verwalten, wenn die Browser der Benutzer keine Cookies unterstützen.

Die Daten der Sitzung werden im Arbeitsspeicher auf dem Webserver abgelegt und mit der Sitzungs-Id assoziiert. Deswegen benötigen Sitzungen immer auch Ressourcen auf dem Webserver. Sie sollten also unter ASP ermöglichen, dass die Webanwender Cookies zulassen, und dafür sorgen, dass der Timeout der Sitzungen nicht allzu hoch angesetzt ist.

Das Problem mit abgelaufenen Sitzungen

Wenn ein Benutzer bei der Arbeit mit einer Webanwendung zu lange nichts macht (z. B. zwischenzeitlich Mittagspause macht), läuft die Sitzung normalerweise ab. Falls Sie im `Session`-Objekt irgendeinen Status speichern (z. B. den Inhalt des aktuellen Warenkorbs in einem Online-Shop), kann das zum Problem werden. Das `Session`-Objekt wird ja schließlich komplett zerstört, wenn die Sitzung abläuft. Wenn Sie nichts weiter programmieren, arbeitet der Anwender nach seiner

Pause vielleicht weiter, erzeugt damit eine neue Sitzung und wundert sich darüber, dass seine Daten zwischenzeitlich verloren gegangen sind. In speziellen Fällen erzeugen abgelaufene Sitzungen sogar Laufzeitfehler. Wenn Sie z.B. einem Benutzer das sequentielle Durchgehen einer Datenbanktabelle erlauben wollen, müssen Sie ein passendes Datenbankobjekt, z.B. ein `Recordset` bei ADO, im `Session`-Objekt verwalten. Wird das Objekt nicht im Startereignis der Sitzung, sondern über eine ASP-Seite erzeugt (was üblich ist), erzeugt die Script-Engine beim Zugriff auf das nicht erzeugte Objekt einen Laufzeitfehler.

Eine einfache Lösung dieses Problems ist, den Anwender beim Start einer Sitzung zur Startseite der Anwendung umzuleiten. Da das Startereignis immer aufgerufen wird, wenn der Anwender auf ein beliebiges Dokument der Anwendung zugreift, funktioniert dieser »Trick«. Der Anwender erkennt dann zwar nicht, warum er umgeleitet wurde, aber wenigstens, dass irgendetwas passiert ist:

```
<script language="VBScript" runat="Server">
 Sub Session_OnStart()
  ' Anwender zur Startseite umleiten
  Response.Redirect "/Shop/Default.asp"
 End Sub
</script>
```

Eigentlich sollten Sie dem Anwender mitteilen, dass seine letzte Sitzung abgelaufen war, wenn er später noch einmal auf die Anwendung zugreift, so wie dies in manchen Webanwendungen der Fall ist. In Anwendungen, in die der Benutzer sich einloggen muss, ist das auch kein Problem. Der Benutzer-Loginname wird dazu nach dem Login in einer Session-Variable gespeichert, die einfach auf jeder ASP-Seite abgefragt wird. Ist diese Variable nicht vorhanden, wird zu einer ASP-Seite mit Informationen zur abgelaufenen Sitzung umgeleitet. Für Anwendungen, die keinen Login erfordern, habe ich bisher noch keine Lösung gefunden. Die Idee, im Ende-Ereignis der Sitzung ein Cookie zu schreiben (das aussagt, dass die Sitzung korrekt beendet wurde), scheitert daran, dass naturgemäß in diesem Ereignis das `Response`*-Objekt nicht verfügbar ist.*

Das Prinzip der Verwaltung von Statusdaten in einer Datenbank: GUIDS für die Benutzer

Viele Anwendungen umgehen das Sitzungs-Problem, indem sie Statusdaten entweder sofort oder im Ende-Ereignis der Sitzung serverseitig in eine Datenbank oder Datei schreiben. Etwas problematisch dabei ist, dass diese Daten mit dem Benutzer assoziiert werden müssen. Die Sitzungs-Id können Sie dazu nicht verwenden. Der IIS vergibt für jede Sitzung eine separate Id. Frei gewordene Ids werden auch schon einmal an andere Sitzungen vergeben. Die Lösung ist einfach: Verwenden Sie eine GUID (Global Unique Identifier). Eine GUID ist eine weltweit eindeutige Id, die nach einem ausgefeilten Algorithmus erzeugt wird. Neben dem aktuellen, bis auf die Mikrosekunde genauen Datum werden z. B. aktuelle CPU-Registerinhalte und die ID der Netzwerkkarte in diese ID hineingerechnet. Eine typische GUID sieht z. B. so aus:

```
{F2DD04E5-7D23-431B-AF4E-A0A9CC6B0E3E}
```

Diese GUID speichern Sie als Cookie auf dem Rechner des Benutzers und lesen sie beim Start der Sitzung wieder aus. Diese GUID wird dann in der Datenbank als Id für den Benutzer verwaltet. Das folgende Listing zeigt, wie dies programmiert werden kann:

```
<script language="VBScript" runat="Server">

 Sub Session_OnStart()
   ' GUID des Benutzers aus dem Cookie auslesen
   Dim strGUID
   strGUID = Request.Cookies("UserGUID")
   ' Falls keine GUID existiert, eine neue erzeugen
   If strGUID = "" Then
     ' Noch keine GUID gespeichert: Neue GUID erzeugen
     Dim objTypeLib
     Set objTypeLib = CreateObject("Scriptlet.Typelib")
     strGUID = objTypeLib.Guid
     ' In ASP.NET: System.GUID.NewGUID
     ' und als Cookie ablegen
     Response.Cookies("UserGUID") = strGUID
     Response.Cookies("UserGUID").Expires = Now + 365
```

```
  End If

  ' GUID im Session-Objekt speichern
  Session("UserGUID") = strGUID
 End Sub

</script>
```

Die GUID wird im Beispiel einfach über die Scriptlet-Bibliothek erzeugt (die unter Windows 2000 mit dem IIS 5 verfügbar ist, ich weiß allerdings nicht, ob das immer der Fall ist. Zur Not suchen Sie diese Bibliothek bei *search.Microsoft.com/us/dev*). Diesen »Trick« habe ich selbst über *groups.google.com* recherchiert. Dank an den Verfasser.

In ASP.NET sollten Sie zur Erzeugung einer GUID den folgenden Quellcode verwenden:

```
Dim strGUID As String
strGUID = System.Guid.NewGuid().ToString()
```

Sitzungs-Timeout temporär umstellen

Ein häufiges Problem bei der Arbeit mit Sitzungen ist, dass der Timeout für einige Aktionen zu gering eingestellt ist. Den Timeout dann generell hochzusetzen, ist nicht besonders optimal, da jede Sitzung Ressourcen auf dem Webserver beansprucht. Sie können den Timeout aber auch für einzelne Dokumente temporär umstellen. Dazu müssen Sie sich den originalen Timeout zunächst im `OnStart`-Ereignis der Sitzung im `Session`-Objekt selbst merken:

```
<!--Global.asa -->
<script language="VBScript" runat="server">
 Sub Session_OnStart()
  ' Merken des originalen Timeout
  Session("OriginalTimeout") = Session.Timeout
 End Sub
</script>
```

Stellen Sie den Timeout dann auf jeder ASP-Seite individuell ein, wobei Sie auf »normalen« Seiten den originalen Timeout verwenden:

```
<!-- NormaleBearbeitung.asp -->
<%@Language="VBScript"%>
<%
 Option Explicit
 Session.Timeout = Session("OriginalTimeout")
%>
<html>
...
```

Auf Seiten, die eine lange Bearbeitung erfordern, stellen Sie den Timeout dann auf einen passenden Wert (in Minuten) ein:

```
<!-- LangeBearbeitung.asp -->
<%@Language="VBScript"%>
<%
 Option Explicit
 Session.Timeout = 120
%>
<html>
...
```


9.6 Verwalten von Cookies

Sitzungen bieten schon viele Möglichkeiten, Daten sitzungsweit zu speichern. Manchmal müssen Daten aber auch zwischen einzelnen Sitzungen des Anwenders verfügbar sein. Wenn Sie dazu keine Datenbank verwenden wollen oder können (mit dem GUID-Trick, den ich auf Seite 252 vorgestellt habe), verwenden Sie einfach Cookies. Ein Cookie (deutsch: »Plätzchen«!?) ist eine Speichereinheit, die auf dem Rechner des Benutzers abgelegt wird. Es gibt temporäre Cookies, die der Browser nur im Arbeitsspeicher verwaltet (die Sitzungs-Id wird so gespeichert) und persistente Cookies, die auf der Festplatte abgelegt werden. Persistente Cookies besitzen ein Ablaufdatum. Der Browser überprüft beim Start oder beim Beenden, ob Cookies abgelaufen sind, und löscht diese dann automatisch.

Cookies werden von vielen Anwendern verteufelt. Ein früher Artikel in einer bekannten Computerzeitschrift hat fälschlicherweise behauptet, Cookies könnten von beliebigen Internetanwendungen ausgele-

sen werden. Böswillige Anwendungen könnten so z. B. das Surf- und Einkaufverhalten eines Benutzers herausfinden. Das stimmt aber so nicht. Cookies werden immer mindestens mit der Adresse des Webservers assoziiert. Andere Webserver können diese Cookies nicht lesen. Hacker kommen natürlich an die Daten heran, wenn der Rechner nicht geschützt ist, aber das ist ein anderes Thema. Ganz unproblematisch sind Cookies allerdings nicht, einige Internetfirmen arbeiten mit fiesen Tricks. Bei *www.cookiecentral.com* finden Sie weitere Informationen dazu.

Der Benutzer kann die Speicherung von Cookies im Browser verhindern. Einige ältere oder einfache Browser unterstützen vielleicht auch keine Cookies. Sie können also im Internet nie sicher sein, dass der Browser übertragene Cookies auch speichert. Fehler entstehen dadurch nicht. Beim Lesen erhalten Sie einfach einen leeren Eintrag zurück. Sie müssen aber darauf reagieren. Nur in Intranets können Sie Cookies voraussetzen, da Sie den teilhabenden Personen mitteilen können, dass diese für Ihre Anwendung Cookies ermöglichen sollten.

Wie werden Cookies zwischen Webserver und Browser ausgetauscht?

Cookies werden vom Webserver aus einfach im HTTP-Header an den Browser gesendet. Der folgende HTTP-Header aus einem Response beinhaltet zwei Cookies:

```
HTTP/1.1 200 OK.
Server: Microsoft-IIS/5.0.
Date: Thu, 04 Oct 2001 19:35:55 GMT.
Content-Length: 253.
Content-Type: text/html.
Set-Cookie: Nachname=Bayer; expires=Fri, 04-Oct-2002
 20:35:54 GMT; path=/asp-beispiele.
Set-Cookie: Vorname=J%FCrgen; expires=Fri, 04-Oct-2002
 20:35:54 GMT; path=/asp-beispiele.
Cache-control: private.
.
```

Der Browser wertet den Header aus und speichert die Cookies auf dem Rechner. Der Netscape speichert das Cookie des Beispiels z. B. so:

```
Trillian  FALSE  /asp-beispiele FALSE  1033772424
  Nachname  Bayer
Trillian  FALSE  /asp-beispiele  FALSE  1033772424
  Vorname  J%FCrgen
```

Fordert der Browser Dokumente vom Webserver an, sendet er die mit dem Webserver assoziierten Cookies wieder im HTTP-Header zum Server. Cookies, bei denen ein Pfad angegeben ist (wie im Beispiel oben), werden nur gesendet, wenn die Anforderung an diesen Pfad geht.

```
GET /asp-beispiele/CookieDemo.asp HTTP/1.1.
Accept: image/gif, image/x-xbitmap, image/jpeg,
 image/pjpeg, application/msword, */*.
Accept-Language: de.
Accept-Encoding: gzip, deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01;
 Windows NT 5.0).
Host: Trillian.
Connection: Keep-Alive.
Cookie: Nachname=Bayer; Vorname=J%FCrgen.
.
```

Der Vorgang beim Speichern von Cookies ist also eigentlich ganz einfach. Wenn Sie Ihre Cookies einmal anschauen wollen, öffnen Sie für den Netscape die Datei COOKIES.TXT im Benutzer-Ordner von Netscape (normalerweise \PROGRAMME\NETSCAPE\USERS\DEFAULT). Der Internet Explorer speichert Cookies im Ordner \DOKUMENTE UND EINSTELLUNGEN*Benutzername*\COOKIES.

Verwalten von Cookies in ASP

Zur Speicherung von Cookies verwenden Sie die `Cookies`-Auflistung des `Response`-Objekts. In ASP fügen Sie das Cookie folgendermaßen an:

```
Response.Cookies("Vorname") = "Jürgen"
```

Danach sollten Sie noch einige Eigenschaften des Cookie einstellen. Wichtig ist z. B. die Einstellung des Verfallsdatums, wenn das Cookie persistent gespeichert sein soll:

```
With Response.Cookies("Vorname")
   .Expires = Now + 365
End With
```

Im Beispiel wird das Verfallsdatum auf das aktuelle Datum addiert mit einem Jahr gesetzt. Das Cookie ist also ein Jahr gültig.

In ASP.NET verwenden Sie eine etwas andere Technik:

```
Dim cookie As HttpCookie
cookie = New HttpCookie("Vorname", _
   Request.Form("Vorname"))
cookie.Expires = Now.AddDays(365)
Response.Cookies.Add(cookie)
```

Cookies können auch strukturiert gespeichert werden. Die Daten einer Person können Sie z. B. unter einem Eintrag speichern:

```
Response.Cookies("User")("Vorname") = "Jürgen"
Response.Cookies("User")("Nachname") = "Bayer"
Response.Cookies("User").Expires = Now + 365
```

Cookies besitzen die in Tabelle 9.1 dargestellten Eigenschaften.

Eigenschaft	Bedeutung
`Domain`	In dieser lesegeschützten Eigenschaft können Sie eine Domäne angeben, für die das Cookie dann gelten soll. Geben Sie einen vollen Namen an, gilt das Cookie nur für diesen einen Rechner. Sie können aber auch Teildomänen angeben. `Domain="microsoft.com"` bewirkt dann z. B., dass alle Anforderungen an *www.microsoft.com, search.microsoft.com* etc. das Cookie erhalten. Wenn Sie diese Eigenschaft nicht angeben, verwendet ASP den Domänennamen des Webservers bzw. im Intranet den Rechnernamen.
`Expires`	Ablaufdatum des Cookie. Wenn Sie diese Eigenschaft nicht setzen, läuft das Cookie sofort ab, wird also nur temporär im Browser gespeichert.

Eigenschaft	Bedeutung
`HasKeys`	Über diese schreibgeschützte Eigenschaft können Sie überprüfen, ob ein Cookie selbst wieder eine Auflistung speichert.
`Name` (nur ASP.NET)	In ASP.NET können Sie über diese Eigenschaft den Namen des Cookie setzen und lesen.
`Path`	Diese lesegeschützte Eigenschaft definiert den Pfad der Dokumente, die dieses Cookie gesendet bekommen sollen. Wenn Sie nichts eintragen, wird der aktuelle Pfad verwendet. Wenn Sie hier "\" eintragen, wird das Cookie an alle Dokumente des Webservers gesendet.
`Secure`	Diese lesegeschützte Eigenschaft definiert, ob das Cookie sicher ist.

Tabelle 9.1: Die Eigenschaften eines Cookie-Objekts (für ASP.NET nur die wichtigsten)

Das Auslesen von Cookies ist sehr einfach. Verwenden Sie dazu das Request-Objekt. Um herauszufinden, ob ein Cookie existiert, können Sie einfach mit einem Leerstring vergleichen. Das folgende Beispiel leitet den Anwender auf eine Login-Seite um, wenn eines der erwarteten Cookies nicht gefunden wurde (über die Login-Seite werden die Cookies dann gespeichert. Sie finden das komplette Beispiel in den Quellcodes zu diesem Kapitel):

```
If Request.Cookies("Vorname") > "" And _
   Request.Cookies("Nachname") > "" Then
   ' Redirect zum Login
   Response.Redirect "Login.htm"
Else
   Response.Write "<b>Hallo " & _
      Request.Cookies("Vorname") & " " & _
      Request.Cookies("Nachname")
End If
```

In ASP.NET sieht das Ganze wieder etwas anders aus. Hier müssen Sie die `Value`-Eigenschaft abfragen. Bei der Überprüfung auf Existenz müssen Sie das Objekt überprüfen:

```
If (Request.Cookies("Vorname") Is Nothing Or _
   Request.Cookies("Nachname") Is Nothing Then
   ...
Else
   Response.Write("<b>Hallo " & _
      Request.Cookies("Vorname").Value  & " " & _
      Request.Cookies("Nachname").Value)
End If
```

Strukturierte Cookies lesen Sie ähnlich:

```
Response.Write "<p>" & _
   Request.Cookies("User")("Vorname") & " " & _
   Request.Cookies("User")("Nachname")
```

Über die `HasKeys`-Eigenschaft können Sie herausfinden, ob ein Cookie Untereinträge besitzt.

9.7 ASP-Seiten umleiten (Redirect)

In manchen Situationen ist es notwendig, den Anwender auf ein anderes Dokument umzuleiten. Im letzten Beispiel habe ich den Anwender bereits umgeleitet, wenn beim Abruf einer ASP-Seite festgestellt wird, dass der Anwender nicht eingeloggt ist. Sinnvoll ist ein Umleiten aber auch für Fehlermeldungen: Tritt ein Fehler auf, wird zu einer Fehlerausgabe-Seite umgeleitet (so wie es der IIS bereits implizit macht).

Das Umleiten ist eigentlich ganz einfach: `Response.Redirect(`*URL*`)` führt zu einer Umleitung zur angegebenen URL. Der folgende Quellcode leitet den Anwender z. B. zur Nitty-Gritty-Seite um:

```
Response.Redirect "www.nitty-gritty.de"
```

Um Umleitungen zu verstehen, sollten Sie wissen, wie diese funktionieren. Fordert ein Client ein ASP-Dokument an, das die Anforderung umleitet, sendet der IIS einen spezielle Umleitungs-Response an den Client zurück:

```
HTTP/1.1 302 Object moved.
Server: Microsoft-IIS/5.0.
Date: Fri, 05 Oct 2001 00:24:01 GMT.
```

```
Location: Login.htm.
Content-Length: 156.
Content-Type: text/html.
Cache-control: private.
.
<head><title>Objekt verschoben</title></head>
<body><h1>Objekt verschoben</h1>Klicken Sie hierauf, um das Objekt <a
HREF="Login.htm">anzuzeigen</a>.</body>
```

Ist der Browser schlau (also modern), ruft er das im Location-Attribut des HTTP-Headers angegebene Dokument dann automatisch ab. Ältere Browser-Versionen zeigen aber auch schon einmal das HTML-Dokument im Body des Response an.

Aus dieser Technik ergibt sich, dass der Webserver noch keine Daten an den Client gesendet haben darf, bevor eine Umleitung gestartet wird. Wenn Sie das Puffern der Response-Daten abgeschaltet haben (was Sie unter dem IIS 5 explizit machen müssten), erhalten Sie eine Fehlermeldung, wenn Sie vor der Umleitung bereits irgendwelche HTML-Daten erzeugt haben. Der folgende Quellcode erzeugt z. B. den in Abbildung 9.2 dargestellten Fehler:


```
<%@Language="VBScript" %>
<%Option Explicit%>
<% Response.Buffer=False ' Nur zu Demozwecken !%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<title>Cookies speichern</title>
</head>
<body>
<%

 ' Überprüfen, ob die Cookies existieren
 If Request.Cookies("Vorname") = "" Or _
    Request.Cookies("Nachname") = "" Then
' Redirect zum Login
```

```
Response.Redirect "Login.htm"
 Else
    Response.Write "<b>Hallo " & _
  Request.Cookies("Vorname") & " " & _
  Request.Cookies("Nachname")
 End If
```

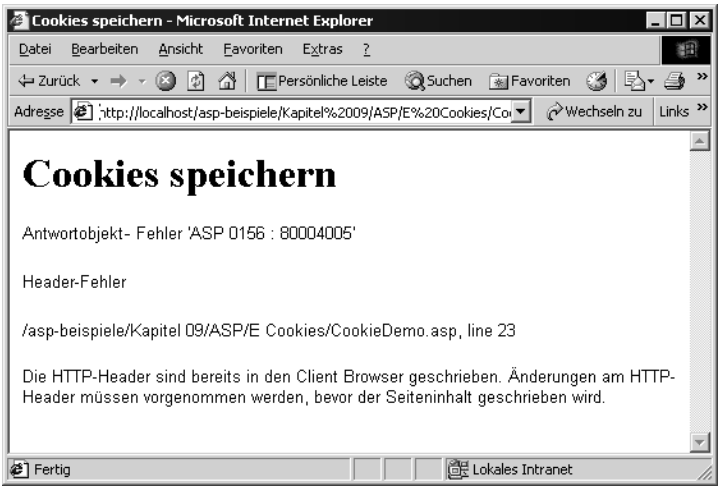


Bild 9.2: Ein Laufzeitfehler bei einer Umleitung mit ausgeschaltetem Puffer

Wenn Sie umleiten, sollten Sie den Puffer also immer eingeschaltet haben (was im IIS 5 ja per Voreinstellung der Fall ist). Normalerweise sollten Sie dann vor der Umleitung noch den Puffer mit `Response.Clear` leeren, im IIS 5 ist das aber scheinbar nicht notwendig (es funktioniert auch einwandfrei ohne Leeren des Puffers).

10 Datenbankzugriff unter ASP

Der Zugriff auf Datenbanken ist unter ASP eines der wichtigsten Themen. Unter ASP verwenden Sie dazu die »Microsoft ActiveX Data Objects Library« (ADO), die mit dem IIS mit installiert wird. ADO ist eine COM-Komponente mit Klassen, über die der Datenbankzugriff relativ einfach, aber auch sehr flexibel ist.

ASP.NET arbeitet dagegen mit ADO.NET. ADO.NET ist ADO-ähnlich, bietet aber wesentlich flexiblere und einfachere Möglichkeiten, mit Datenbanken zu arbeiten. Über das in ASP.NET enthaltene DataGrid-Steuerelement können Sie zum Beispiel mit nur wenigen Zeilen Quellcode den Inhalt einer Datenbankabfrage in HTML darstellen. In ASP haben Sie leider einiges an Arbeit, wenn Sie mit Datenbanken arbeiten, weil Sie alles »von Hand« programmieren müssen.

HiN/WEIS

Unter ASP.NET können Sie zwar auch »von Hand« programmieren, so wie ich es in diesem Kapitel zeige. Dazu können Sie neben ADO.NET sogar auch noch ADO verwenden. Diese Art der Programmierung dürfte unter ASP.NET aber nicht mehr allzu wichtig sein. Die speziellen ASP.NET-Steuerelemente erleichtern die Arbeit mit Datenbanken einfach viel zu sehr. Ich beschreibe in diesem Kapitel deshalb nur die Arbeit mit ADO unter ASP. Kapitel 14 zeigt, wie Sie unter ASP.NET mit Datenbanken arbeiten.

ADO ist sehr vielfältig. Ich beschreibe hier deshalb nur die für die Arbeit mit ASP wesentlichen Dinge. Einige erweiterte Techniken werden in verschiedenen Artikeln beschrieben, die Sie auf der Buchseite bei *www.nitty-gritty.de* finden.

Die in den Beispielen verwendete Datenbank

Um nicht vorauszusetzen, dass Sie den SQL Server installiert haben, verwende ich für die Beispiele dieses Kapitels meist die Access-Datenbank NORDWIND.MDB. Diese Datenbank wird mit Microsoft Access

als Beispiel mitgeliefert und enthält Tabellen für die Bearbeitung von Bestellungen (Artikel, Kunden, Bestellungen etc.). Eine ähnliche Datenbank, die NWIND.MDB, ist Bestandteil von Visual Basic 6. Der SQL Server installiert eine ähnliche Datenbank, die dort allerdings NORTHWIND heißt und englische Tabellen- und Feldnamen verwendet. Da ich nicht voraussetzen kann, dass Sie diese Datenbank besitzen, finden Sie die Access-Version in einer um für uns unnötige Bestandteile (Abfragen, Formulare, Berichte etc.) reduzierten (und damit kleineren) Variante in den Downloads zum Buch.

10.1 Grundlagen

10.1.1 Die ActiveX Data Objects (ADO)

Die ActiveX Data Objects, die Sie zum Zugriff auf Datenbanken verwenden, werden in der Version 2.7 automatisch bei der Installation des IIS mit installiert. ADO ist eine COM-Komponente, die verschiedene Klassen für die Arbeit mit beliebigen Datenquellen enthält. Über diese Klassen können Sie nicht nur auf Datenbanken, sondern auch auf Datenquellen, wie zum Beispiel Textdateien, E-Mail-Systeme, Index- oder LDAP-Server zugreifen. Die einzige Voraussetzung dazu ist, dass Sie einen passenden Data-Provider besitzen. Der Data-Provider ist das Bindeglied zwischen ADO und der Datenquelle. Data-Provider werden häufig vom Hersteller des Datensystems entwickelt und frei zur Verfügung gestellt. Wenn Sie zum Beispiel auf eine Oracle-Datenbank zugreifen wollen, können Sie den aktuellen Oracle-Provider von *www.oracle.com* downloaden. ADO liefert jedoch bereits einige Provider für die wichtigsten Datensysteme mit. Falls Sie für eine Datenbank einmal keinen Provider finden, können Sie immer noch über ODBC Verbindung zur Datenbank aufnehmen, indem Sie den ODBC-Provider verwenden.

10.1.2 Das Prinzip des Datenzugriffs unter ASP

In ASP-Anwendungen müssen Sie den Datenzugriff prinzipiell »von Hand« programmieren. Das bedeutet, das Sie Daten, die Sie abgefragt haben, mit `Response.Write` in ein ASP-Dokument schreiben. Wenn Sie das Editieren oder Anfügen von Daten ermöglichen wollen, müssen Sie ein HTML-Formular zur Verfügung stellen und die einge-

gebenen Daten mit Hilfe von ADO in die Datenquelle schreiben. Das Anbinden einer Datenquelle an spezielle Steuerelemente, wie es in Visual Basic oder Borland Delphi möglich ist, ist in ASP prinzipiell nicht möglich. Für diejenigen, die diese Technik nicht kennen: Anbinden bedeutet, dass Sie Daten über ein Datenzugriffsobjekt (bei ADO ist das ein `Recordset`-Objekt) abfragen und dieses Objekt dann an einzelne Steuerelemente anbinden, die dann ganz automatisch die Anzeige der gespeicherten Daten übernehmen. Visual Interdev bietet allerdings spezielle Steuerelemente, die ein solches Anbinden simulieren. Dort erzeugen Sie eine Datenbankverbindung und eine Abfrage mit Hilfe von Dialogen und binden die abgefragten Daten ähnlich wie Visual Basic an so genannte Entwurfszeit-Steuerelemente an. Diese Steuerelemente werden von Visual Interdev automatisch in HTML-Steuerelemente umgesetzt. Die Anbindung der Daten an diese Steuerelemente wird von Visual Interdev dann im Hintergrund, über automatisch erzeugte ASP-Skripte realisiert. Im Prinzip erzeugt Visual Interdev denselben Quellcode, den auch wir erzeugen, wenn wir Datenquellen direkt in ASP bearbeiten. Visual Interdev beschreibe ich nicht, weil es den Rahmen dieses Buchs sprengen würde (schöne Phrase für »Ich habe zu wenig Platz« ...). Dieses Kapitel behandelt also den Datenzugriff an der Basis.

In Kapitel 14 beschreibe ich, wie Sie mit ASP.NET Datenquellen anzeigen und bearbeiten. In ASP.NET können Sie Datenquellen sehr einfach an verschiedene Steuerelemente anbinden und damit erreichen, dass die Daten automatisch angezeigt werden.

10.1.3 Die ADO-Konstanten

ADO-Objekte verwenden viele Argumente, die normalerweise über symbolische Konstanten angegeben werden. In ASP sind diese Konstanten aber zunächst nicht vorhanden. Um nicht die undurchsichtigen Zahlwerte der ADO-Konstanten verwenden zu müssen (wer weiß schon, welchen Wert die Konstante `adOpenStatic` besitzt), binden Sie idealerweise eine Include-Datei ein, die diese Konstanten definiert. Diese Datei mit dem Namen ADOVBS.INC finden Sie normalerweise im Ordner \Programme\Gemeinsame Dateien\System\ado. Kopieren Sie diese Daten in einen Ordner unterhalb des IIS-Stammordners und

binden Sie diese dann mit #Include ein. Ich kopiere die Datei grundsätzlich immer in den Ordner der Webanwendung, damit dort alle Dateien zusammen gespeichert sind:

```
<!-- #Include file="adovbs.inc" -->
```

10.2 Verbindung aufnehmen: Das Connection-Objekt

Jeder Datenzugriff benötigt eine Verbindung zur Datenquelle. Diese können Sie prinzipiell für jede Datenabfrage separat aufbauen (indem Sie dem `Recordset`-Objekt die Verbindungsinformationen beim Öffnen der Abfrage jeweils mitgeben). Im Idealfall bauen Sie die Verbindung aber über ein Objekt der Klasse `Connection` auf. Damit besitzen Sie mehr Möglichkeiten. Transaktionen können zum Beispiel nur über ein `Connection`-Objekt gehandhabt werden. Dieses Objekt geben Sie dann beim Öffnen von Abfragen für die Verbindung an.

Erzeugen Sie also ein Objekt der Klasse `Connection`. Das Objekt erzeugen Sie mit der unter ASP üblich Syntax. Als Komponentennamen geben Sie aber nicht, wie anzunehmen wäre, »ADO« an, sondern »ADODB«:

```
Dim con
Set con = Server.CreateObject("ADODB.Connection")
```

Die Verbindung öffnen Sie dann mit der `Open`-Methode

connection.Open [*ConnectionString*] [, *UserID*] [, *Password*]

Bis auf die Zusammensetzung des Verbindungsstrings im Argument *ConnectionString* ist das Öffnen der Verbindung recht unproblematisch. Der Verbindungsstring besteht aus mehreren Argumenten, die mit einem Semikolon voneinander getrennt sind. Da jeder Provider unterschiedliche Argumente verwendet, sieht der Verbindungsstring für jeden Provider auch ganz unterschiedlich aus. Das einzige Argument, das immer vorkommt, ist das Argument *Provider* (in dem Sie den zu verwendenden Provider angeben, aber das sagt ja schon der Name des Arguments). Wenn Sie das Verbindungsstring-Argument *Provider* weglassen, verwendet ADO immer den ODBC-Provider.

In den Argumenten *UserId* und *Password* der `Open`-Methode geben Sie für geschützte Datenbanken den Benutzernamen und dessen Passwort an, der für die Datenbankoperationen verwendet werden soll. Alternativ können Sie diese Infos auch in den Argumenten *UID* und *PWD* im Verbindungsstring angeben, aber wer weiß, ob alle Provider diese Verbindungsstring-Argumente unterstützen. Bei einer Verbindung zu einer SQL Server-Datenbank müssen Sie die User-ID und das Passwort angeben:

```
con.Open "Provider=SQLOLEDB;Data Source=Zaphod;" & _
         "Initial catalog=Pubs", "sa", "bandit"
```

In diesem Beispiel ist »Zaphod« der Name des SQL Servers, »Pubs« der Name der Datenbank, »sa« der Benutzername und »bandit« das Kennwort des Benutzers.

Wenn beim Öffnen kein Fehler auftritt, ist die Verbindung zur Datenbank hergestellt. In der Praxis sollten Sie natürlich immer auf mögliche Fehler reagieren, wie ich es bereits in Kapitel 7 beschrieben habe.

Schließen können Sie eine Verbindung dann über die `Close`-Methode.

Wo öffnen und wie lange geöffnet halten?

Eine wichtige Frage bei der Herstellung von Datenbankverbindungen ist, wo diese geöffnet werden und wie lange die Verbindung geöffnet bleiben soll. In ASP haben Sie dazu eigentlich nur zwei Möglichkeiten: Entweder öffnen Sie die Verbindung am Anfang jeder ASP-Seite und schließen diese direkt am Ende der Seite. Dazu benötigen Sie eine Connection-Variable, die nur auf der Seite gilt. Oder Sie erzeugen und öffnen die Verbindung im Startereignis der Sitzung, speichern das `Connection`-Objekt in einer `Session`-Variablen und schließen die Verbindung dann im Ende-Ereignis der Sitzung. Die zweite Variante besitzt wenig Vorteile, aber einen gravierenden Nachteil: Die Verbindung bleibt so lange geöffnet, wie die Sitzung aktiv ist. Damit werden auf dem Server Ressourcen verwendet, die in der ASP-Anwendung wahrscheinlich nur sehr selten wirklich benötigt werden. Greifen sehr viele Benutzer gleichzeitig auf die Anwendung zu, kann das zum Problem werden. Ein anderer Nachteil ist, dass auf dem Server genügend Lizenzen für Datenbankverbindungen zur Verfügung stehen müssen. Moderne Systeme verwenden aber meist eine eher

geringe Anzahl Lizenzen und arbeiten so, dass Verbindungen nur bei Bedarf aufgebaut werden. So können viele gleichzeitige Benutzer mit nur wenigen lizensierten Verbindungen bedient werden. Ein ADO-Feature, das Connection-Pooling, unterstützt diese Technik: Eine geschlossene Verbindung bleibt unter ADO noch für eine Minute intern geöffnet, so dass ein Client, der eine Verbindung anfordert, diese sehr schnell erhält. Wenn Sie einmal bei Ihren ADO-Versuchen darauf achten, wie viel Zeit der erste Verbindungsaufbau im Vergleich zu folgenden benötigt, werden Sie erkennen, dass das permanente Schließen und Wieder-Öffnen einer Verbindung eigentlich gar keine Zeit beansprucht. In den Beispielen dieses Kapitels öffne ich die Verbindung also immer im Kopf der ASP-Seite und schließe diese im Fuß.

Ein Tipp für das Öffnen und Schließen

Programmieren Sie das Öffnen und Schließen idealerweise in einer Include-Datei, die Sie mit `<!-- #include file="Connection.inc" -->` in alle ASP-Dokumente einbinden, die einen Datenbankzugriff benötigen.

Verbindung erzeugen: The easy way

Normalerweise bin ich kein Freund von Tools, die komplexe Dinge wie das Öffnen einer Datenbankverbindung erleichtern. Diese Tools verstecken meist zu viel und geben Ihnen wenig Möglichkeiten, den erzeugten Programmcode zu beeinflussen. ADO enthält aber einen genialen Dialog zur Erzeugung einer Verbindung, den Sie sogar unter ASP nutzen können. Mit Hilfe dieses Dialogs erzeugen Sie eine so genannte Datenverknüpfungsdatei mit der Endung .UDL (Universal Data Link), die die Verbindungsinformationen speichert. Diese Datei geben Sie beim Öffnen der Verbindung an. So ganz nebenbei sehen Sie im Verbindungsdialog auch noch, welche Provider auf Ihrem System installiert sind und welche Attribute für den jeweiligen Provider möglich sind. Außerdem ermöglichen Sie dem Webadministrator über eine Datenverknüpfungsdatei, die Verbindungsinformationen auch nachträglich sehr einfach einzustellen. Also, warum schwer, wenn es auch einfach geht.

Wenn Sie trotzdem wissen wollen, wie eine Verbindung zu den verschiedenen Standard-Datenbanken von Hand aufgebaut wird, finden Sie auf meiner Buchseite einen Artikel »ADO-Verbindungen«, der diese beschreibt.

Eine Datenverknüpfungsdatei erzeugen Sie einfach, indem Sie in einem beliebigen Ordner der Festplatte eine Textdatei erzeugen und dieser Datei die Endung .UDL vergeben. Wenn Sie diese Datei dann öffnen, zeigt ADO zur Bearbeitung den Verbindungs-Dialog an.

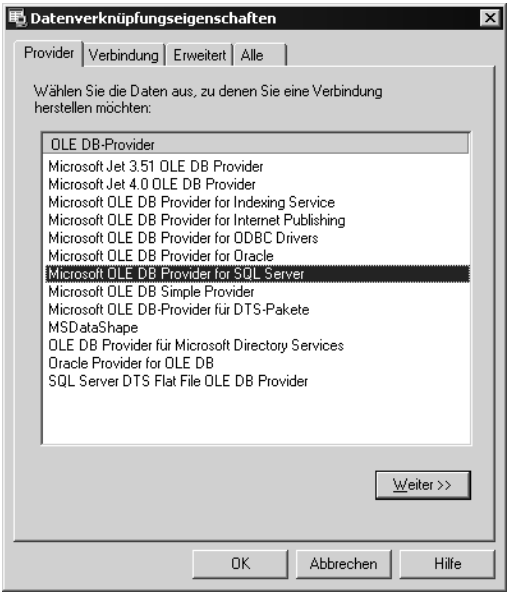


Bild 10.1: Der Dialog zur Einstellung der Verbindungsinformationen

Nachdem Sie im ersten Register den Provider ausgewählt haben, stellen Sie im Register VERBINDUNG die Verbindungsinformationen ein. Dieses Register wird dynamisch erzeugt und sieht bei den verschiedenen Providern ganz unterschiedlich aus. Abbildung 10.2 zeigt die möglichen Einstellungen für den SQL Server.

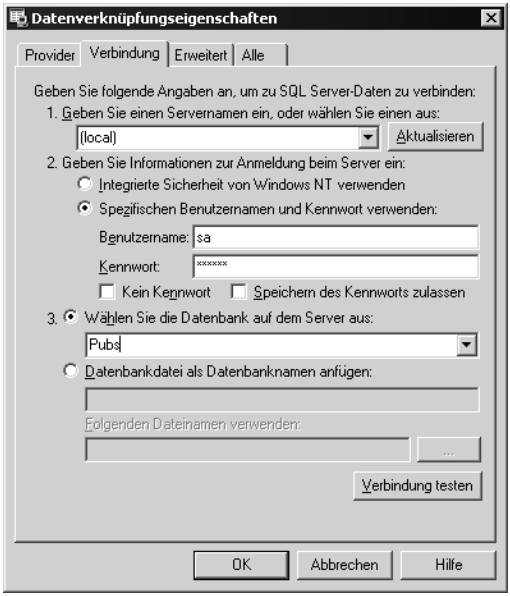


Bild 10.2: Die Verbindungseigenschaften des SQL Servers

Der Benutzername und das Kennwort werden zunächst nur dazu verwendet, für den Dialog die Datenbanken des SQL Servers auszulesen. Nur wenn Sie die Option SPEICHERN DES KENNWORTS ZULASSEN einschalten, werden diese Informationen auch in der Datenverknüpfungsdatei gespeichert. Da ein Kennwort in ASP sowieso nur schlecht geschützt werden kann, können Sie dieses auch gleich in der Datenverknüpfungsdatei angeben. Ist das Kennwort nicht in der Datei gespeichert, müssen Sie den Benutzernamen und das Kennwort beim Öffnen der Verbindung angeben.

Wenn die Datenverknüpfungsdatei dann gespeichert ist, geben Sie diese beim Öffnen der Verbindung im Verbindungsstring über "File Name=" an. Da Sie dazu keine relativen Pfadangaben verwenden können, müssen Sie den Pfad zusammensetzen. Wenn die Datei im Stammordner der ASP-Anwendung gespeichert ist (was ich empfehle), ermitteln Sie den Stammordner der ASP-Anwendung über `Server.MapPath(".")`:

```
con.Open "File Name=" & _
   Server.MapPath(".") & "\Verbindung.udl"
```

10.3 Daten bearbeiten: Das Recordset-Objekt

Das `Recordset`-Objekt ist das komplexeste und wichtigste ADO-Objekt. Über dieses Objekt fragen Sie Daten ab und können diese auch bearbeiten (editieren und anfügen). `Recordset`-Objekte werden mit einer Abfrage geöffnet, die in der Abfragesprache der Datenquelle formuliert. Bei Datenbanken ist das fast immer SQL.

Beim Öffnen von Datenbankabfragen reicht die Angabe des Tabellennamens aus. ADO fragt dann einfach alle Datensätze ab. Für gezielte Abfragen benötigen Sie SQL-Kenntnisse. SQL wird in diesem Buch nicht beschrieben. Auf der Website zum Buch finden Sie aber einen ausführlichen Artikel zum Standard-SQL und zum SQL des SQL Servers.

Recordset-Objekte können Sie ganz einfach über die `Execute`-Methode eines `Connection`-Objekts erzeugen und öffnen. Das folgende Beispiel fragt alle Datensätze der Tabelle »Artikel« (der Nordwind-Datenbank) ab:

```
Dim rstArtikel
Set rstArtikel = con.Execute("Artikel")
```

Ein so erzeugtes Recordset ist schreibgeschützt und kann nur einmal von vorne nach hinten durchlaufen werden. Wenn Sie das Recordset auch dazu verwenden wollen, Daten zu aktualisieren, zu löschen oder anzufügen, müssen Sie dieses separat erzeugen und öffnen. Dabei geben Sie dann den Cursor und den Sperrmechanismus an, den das Datenbanksystem verwenden soll: Das folgende Beispiel erzeugt und öffnet ein Recordset mit einem statischen Cursor (der das Editieren erlaubt) und einem »positiven« Sperrmechanismus:

```
Dim rstKategorien
Set rstKategorien = Server.CreateObject("ADODB.Recordset")
rstKategorien.Open "Kategorien", con, _
  adOpenStatic, adLockOptimistic
```

Im ersten Argument der `Open`-Methode geben Sie die Abfrage an. Im zweiten Argument übergeben Sie die `Connection`-Variable. Das dritte Argument steuert den Cursor, das vierte den Sperrmechanismus. Ich möchte das komplexe Thema »Cursor« hier nicht intensiv behandeln. In den zusätzlichen Kapiteln zu diesem Buch finden Sie einen Artikel »ADO-Cursor und Sperrmechanismen«, der den Umgang mit Cursorn und Sperrmechanismen ausführlich beschreibt. Tabelle 10.1 gibt einen Überblick über die in ADO möglichen Cursor.

Konstante	Beschreibung
`adOpenForwardOnly`	Vorwärts-Cursor. Voreinstellung. Das Recordset kann nur einmal Datensatz für Datensatz durchgeblättert werden.
`adOpenStatic`	Statischer Cursor. Die Datensatzliste wird komplett (mit allen Feldern) im Arbeitsspeicher verwaltet. Änderungen anderer Benutzer an den Daten sind nicht sichtbar.
`adOpenKeyset`	Schlüsselgruppen-Cursor. ADO verwaltet nur Schlüsselwerte für die Datensätze im Speicher. Änderungen anderer Benutzer an den Daten sind sichtbar.
`adOpenDynamic`	Dynamischer Cursor. Wie beim Schlüsselgruppen-Cursor sind Änderungen, aber auch neue und gelöschte Datensätze und Änderungen der Sortierung durch andere Benutzer sichtbar.

Tabelle 10.1: Die ADO-Konstanten des Cursor-Typs

Über das vierte Argument steuern Sie, wie bereits gesagt, den Sperrmechanismus. Dieser ist nur dann wichtig, wenn Sie auch Daten in das Recordset schreiben oder daraus löschen wollen. In diesem Fall geben Sie entweder den Wert `adLockPessimistic` oder `adLockOptimistic` an. Beim pessimistischen Sperren sperrt die Datenbank den Datensatz, sobald der erste Wert in das Recordsets geschrieben wird, und gibt den Datensatz erst wieder frei, wenn die `Update`-Methode aufgerufen wird (die den Datensatz dann in der Datenbank aktualisiert). Beim optimistischen Sperren wird nur für den kurzen Zeitpunkt des Aufrufs der `Update`-Methode gesperrt. Welche dieser beiden Methoden Sie verwenden, macht in ASP keinen großen Unterschied. In

ASP bleiben Recordsets normalerweise nie über einen längeren Zeitraum zum Bearbeiten geöffnet, sondern werden innerhalb einer ASP-Seite geöffnet, beschrieben und direkt aktualisiert (falls Sie dazu überhaupt ein Recordset verwenden, siehe bei »Daten anfügen, editieren und löschen« auf Seite 277). Tabelle 10.2 zeigt die möglichen Konstanten für den Sperrmechanismus

Konstante	Sperroption
`adLockReadOnly`	Read Only Sperren. Default: Die Daten können nur gelesen werden.
`adLockPessimistic`	Pessimistisches Sperren: Ein zum Bearbeiten geöffneter Datensatz wird für andere Benutzer zum Bearbeiten gesperrt.
`adLockOptimistic`	Optimistisches Sperren: Ein zum Bearbeiten geöffneter Datensatz wird für andere Benutzer zum Bearbeiten nur für den Augenblick der `Update`-Methode gesperrt.
`adLockBatchOptimistic`	Optimistische Batch-Updates: Wird für Stapelaktualisierungen benötigt. Stapelaktualisierungen werden nicht in diesem Buch, aber auf der Website beschrieben.

Tabelle 10.2: Die Sperroptionen beim Öffnen eines Recordsets

Normalerweise werden Sie ein Recordset mit einer SQL-Abfrage öffnen und ein `Connection`-Objekt als Verbindung angeben:

```
Dim rst
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open "SELECT * FROM Artikel", con, _
   adOpenForwardOnly, adLockReadOnly
```

10.3.1 Bewegen im Recordset und Zugriff auf die Daten

Wenn Sie ein Recordset geöffnet haben, steht der Cursor auf dem ersten Datensatz (sofern das Recordset nicht leer ist). Sie können den Cursor dann mit einigen Move-Methoden bewegen. `MoveNext` setzt den Cursor zum Beispiel auf den nächsten Datensatz. Informa-

tionen über die aktuelle Position und die Anzahl der Datensätze erhalten Sie über die Eigenschaften `BOF` (Bottom Of File), `EOF` (End Of File), `AbsolutePosition` und `RecordCount`. Über die `Fields`-Auflistung erhalten Sie Zugriff auf die einzelnen Felder des Datensatzes. `Fields` ist eine normale Auflistung und besitzt deswegen die Methode `Item`, über die Sie Zugriff auf die Felder erhalten. Wie bei Auflistungen üblich, können Sie einen Integer-Index oder einen String-Schlüssel übergeben. Ein String-Schlüssel bezeichnet hierbei ganz einfach den Namen des Felds im Recordset. `Item` gibt ein Objekt der Klasse `Field` zurück. Dieses besitzt dann auch wieder Eigenschaften (klar, es ist ja auch ein Objekt ...). Die wichtigste Eigenschaft ist `Value`, die den im Feld gespeicherten Wert verwaltet. Eine andere wichtige Eigenschaft ist zum Beispiel `Type`, über die Sie den Datentyp des Feldes ermitteln können.

Das Ganze mag etwas verwirrend sein, wenn Sie aber einmal damit gearbeitet haben, ist es eigentlich ganz einfach. Das folgende Beispiel zeigt, wie Sie die Artikel-Tabelle der Nordwind-Datenbank sequenziell durchgehen und den Inhalt der Felder »Artikelname« und »Einzelpreis« im ASP-Dokument ausgeben:


```
Dim rstArtikel
Set rstArtikel = con.Execute("Artikel")

Response.Write "<h2>Unsere Artikel</h2>"

' Durchgehen des Recordset
Do While rstArtikel.EOF = False
   ' Felderwerte in Variablen ablegen
   Dim strArtikelName, curArtikelPreis
   strArtikelName = _
      rstArtikel.Fields("Artikelname").Value & ""
   curArtikelPreis = _
      FormatCurrency(rstArtikel.Fields("Einzelpreis"), 2)
   ' und ausgeben
   Response.Write  strArtikelName & " " & _
      curArtikelPreis & "<br>"
```

```
    ' Auf den nächsten Datensatz stellen
    rstArtikel.MoveNext
Loop

' Recordset schließen
rstArtikel.Close
```

Dieses Beispiel verwendet einen wichtigen »Trick«: Beim Auslesen des Felds »Artikelname« addiere ich einen Leerstring zum Feldwert. In leeren Datenbankfeldern ist nämlich normalerweise der Wert `Null` *gespeichert. Wenn Sie diesen Wert in Ausdrücken verwenden, resultiert daraus meist ein Laufzeitfehler. Die Addition mit dem Leerstring konvertiert* `Null` *in einen Leerstring und vermeidet den Laufzeitfehler einfach. Der Trick gilt aber nur für Felder, deren Wert Sie als Zeichenkette auswerten wollen. Wollen Sie mit nummerischen Feldern rechnen, müssen Sie stattdessen immer mit* `IsNull` *oder* `IsNumeric` *überprüfen, ob das Feld eventuell leer ist.*

Tabelle 10.3 zeigt die Methoden, über die Sie den Cursor bewegen können. In Tabelle 10.4 finden Sie die wichtigsten Eigenschaften.

Methode	Beschreibung
`MoveFirst`	Stellt den Cursor auf den ersten Datensatz
`MoveNext`	Stellt den Cursor auf den nächsten Datensatz
`MovePrevious`	Stellt den Cursor auf den vorherigen Datensatz
`MoveLast`	Stellt den Cursor auf den letzten Datensatz
`Move(`*NumRecords*, [*Start*]`)`	Stellt den Cursor auf einen bestimmten Datensatz. In *NumRecords* können Sie die Anzahl der zu überspringenden Datensätze angeben. Da hier auch negative Werte möglich sind, können Sie auch rückwärts springen. In *Start* können Sie den Datensatz angeben, ab dem gesprungen werden soll. Geben Sie hier nichts an, wird ab dem aktuellen Datensatz gesprungen.

Tabelle 10.3: Die Bewegungs-Methoden des ADO-Recordset-Objekts

Eigenschaft	Beschreibung
AbsolutePosition	AbsolutePosition enthält die absolute Position des Cursors als Ganzzahl. Stehen Sie z. B. auf dem 10. Datensatz, so enthält AbsolutePosition den Wert 10.
BOF	BOF (Bottom Of File) ist True, wenn der Cursor *vor* (nicht *auf*!) dem ersten Datensatz steht. Steht der Cursor auf BOF, können Sie den Datensatz nicht bearbeiten (weil keiner aktuell ist) und auch nicht mit MovePrevious auf den vorherigen Datensatz wechseln. Der Cursor steht auf BOF, wenn die MovePrevious-Methode auf dem ersten Datensatz angewendet wurde oder wenn das Recordset leer ist.
Bookmark	Diese Eigenschaft verwaltet das Lesezeichen des aktuellen Datensatzes als Variant. Sie können das Lesezeichen lesen und natürlich auch wieder setzen, um zu einem gemerkten Datensatz zurückzuspringen. Der Cursor und der Provider müssen Lesezeichen unterstützen, was nicht immer der Fall ist.
EOF	EOF (End Of File) ist True, wenn der Cursor *hinter* (nicht *auf*!) dem letzten Datensatz steht. Steht der Cursor auf EOF, können Sie wie bei BOF den Datensatz nicht bearbeiten und auch nicht mit MoveNext auf den nächsten Datensatz wechseln. Der Cursor steht auf EOF, wenn die MoveNext-Methode auf den letzten Datensatz angewendet wurde oder wenn das Recordset leer ist.
Fields	Über die Fields-Auflistung erreichen Sie die einzelnen Felder des aktuellen Datensatzes. Die Eigenschaft Value der über Field referenzierten Field-Objekte verwenden Sie, um den gespeicherten Wert zu lesen oder zu schreiben.
RecordCount	RecordCount enthält die Anzahl der Datensätze in der aktuellen Datensatzliste. Beachten Sie, dass RecordCount bei verschiedenen Cursortypen erst aktualisiert wird, wenn Sie mit MoveLast mindestens einmal auf den letzten Datensatz gesprungen sind. Oft enthält RecordCount nach dem Öffnen eines Recordsets zunächst den Wert 0.

Tabelle 10.4: Die wichtigsten Eigenschaften des ADO-Recordset-Objekts

10.3.2 Daten anfügen, editieren und löschen

Das Lesen von Daten reicht häufig nicht aus. Viele Anwendungen müssen dem Anwender auch die Möglichkeit geben, neue Datensätze anzufügen, vorhandene zu Editieren oder zu löschen. Das können Sie natürlich auch alles mit ADO programmieren. Dabei stehen Ihnen grundsätzlich zwei Möglichkeiten zur Verfügung: Einmal können Sie dazu ein Recordset-Objekt verwenden. Zum anderen können Sie einfach einen passenden SQL-Befehl über die `Execute`-Methode ausführen. Die direkte Ausführung eines SQL-Befehls ist immer effizienter als die Verwendung eines Recordsets. Um hier nicht zu viel zu theoretisieren, zeige ich einfach, wie Sie zum Beispiel eine neuen Artikel in die Artikeltabelle schreiben:

```
' Artikel mit SQL anfügen
con.Execute "Insert Into Artikel " & _
   "(Artikelname, [Kategorie-Nr], Einzelpreis) " & _
   "VALUES ('Flour Tortillas', 5, 2.5)

' Artikel über ein Recordset anfügen
Dim rst
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open "Artikel", con, adOpenStatic, adLockOptimistic
rst.AddNew
rst.Fields.Item("Artikelname").Value = "Flour Tortillas"
rst.Fields.Item("Kategorie-Nr").Value = 5
rst.Fields.Item("Einzelpreis").Value = 2.5
rst.Update
rst.Close
```

An Stelle der konstanten Wertangaben müssen Sie in der Praxis natürlich variable Werte einsetzen, z. B. die Daten, die ein Benutzer in einem Formular eingegeben hat. Die Verknüpfung solcher Eingaben mit SQL-Befehlen ist manchmal nicht ganz einfach. Das folgende Beispiel (das Sie komplett in den Beispieldateien dieses Kapitels finden), schreibt einen neuen Artikel, dessen Daten in einem HTML-Formular eingegeben werden, in die Datenbank:

```
If Request.Form("Artikelname") > "" And _
   IsNumeric(Request.Form("Kategorie")) And _
```

```
   IsNumeric(Request.Form("Einzelpreis")) Then
   ' SQL-Befehl zusammensetzen
   Dim strSQL
   strSQL = "Insert Into Artikel (Artikelname, " & _
      "[Kategorie-Nr], Einzelpreis) VALUES (" & _
      "'" & Request.Form("Artikelname") & "', " & _
      Request.Form("Kategorie") & ", " & _
      Replace(Request.Form("Einzelpreis"), ",", ".") & ")"
   ' Datensatz speichern
   On Error Resume Next
   con.Execute strSQL
   If Err.Number <> 0 Then
      Response.Write "<b>Fehler beim Anfügen: " & _
         Err.Description & "</b>"
   Else
      Response.Write "Artikel wurde erfolgreich " & _
         "angefügt."
   End If
   On Error Goto 0
End If
```


Das Beispiel zeigt einige, beim Erzeugen von SQL-Befehlen wichtige Dinge:

- Die Eingaben sollten vor dem Anfügen des Datensatzes natürlich auf Gültigkeit überprüft werden.
- Sie sollten eine Fehlerbehandlung vorsehen für den Fall, dass die Daten trotz Überprüfung doch nicht geschrieben werden können. Dies kann zum Beispiel dann der Fall sein, wenn der Anwender einen Artikelnamen eingibt, der einfach zu lang ist.
- Die Daten von Textfeldern müssen innerhalb des SQL-Befehls in Apostrophe eingefügt werden.
- Nummerische Werte, wie im Beispiel der Einzelpreis, müssen meist in der englischen Schreibweise angegeben werden. Da ein Zahlwert von VBScript automatisch so formatiert wird, wie es in der Ländereinstellung des Computers festgelegt ist, müssen Sie VBScript ein wenig austricksen. Ich ersetze dazu, wie im Beispiel, über die `Replace`-Funktion einfach eingegebene Kommata durch

den Punkt. In Deutschland funktioniert dies, bei englischen Webservern kann das Probleme geben, wenn der Benutzer das englische Tausender-Trennzeichen verwendet (das ja ein Komma ist).

- Datumswerte werden in einer datenbankspezifischen Form angegeben. Die wenigsten Datenbanksysteme unterstützen das ISO-Format, bei dem ein Datum in der Form 'jjjj-mm-tt' angegeben wird. Access verlangt ein Datum z. B. in der englischen Schreibweise, eingefügt in »#«: #31/12/2099#.

Die zweite Variante mit einem Recordset ist wesentlich einfacher, wenn auch nicht unbedingt kürzer. Hier müssen Sie sich nicht um die Formatierung der Daten kümmern. Beim Schreiben in die Felder muss lediglich der Datentyp passen:

```
Dim rst
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open "SELECT * FROM Artikel WHERE 1 = 2",
   con, adOpenStatic, adLockOptimistic
' Anmerkung: Die Abfrage verwendet die Bedingung 1 = 2,
' damit das Recordset keine Daten enthält. Es soll ja
' schließlich nur ein Datensatz angefügt werden.
rst.AddNew
rst.Fields.Item("Artikelname").Value = _
   Request.Form("Artikelname")
rst.Fields.Item("Kategorie-Nr").Value = _
   Request.Form("Kategorie")
rst.Fields.Item("Einzelpreis").Value =
   Request.Form("Einzelpreis")
On Error Resume Next
rst.Update
If Err.Number <> 0 Then
   Response.Write "<b>Fehler beim Anfügen: " & _
      Err.Description & "</b>"
Else
   Response.Write "Artikel wurde erfolgreich angefügt."
End If
On Error Goto 0
rst.Close
```

Sauber ist dieser Quellcode allerdings nicht. Korrekterweise sollten Sie beim Schreiben der Werte in das Recordset separate Fehlerbehandlungen vorsehen, da dort auch Laufzeitfehler auftreten können. Dadurch wäre das Beispiel aber ziemlich unübersichtlich geworden.

Welche der beiden Varianten weniger Arbeit verursacht, hängt davon ab, ob das Recordset bereits geöffnet ist und ob Sie über genügend SQL-Kenntnisse verfügen. Die Recordset-Variante erfordert gar keine SQL-Kenntnisse, ist dafür aber nicht ganz so performant wie die SQL-Variante. Schließlich muss dazu erst ein Recordset geöffnet werden. Das Beispiel macht dabei auch noch einen klassischen Fehler: Es fragt alle Datensätze ab, obwohl ja nur ein Datensatz angefügt werden soll. Sie können dies verhindern, indem Sie eine Abfrage verwenden, die bei keinem Datensatz wahr wird (z. B. »SELECT * FROM Artikel WHERE 1 = 2«). Dann ist die Performance eigentlich schon sehr gut. Bei der Ausführung der `Update`-Methode sendet ADO übrigens auch einen SQL-Befehl zur Datenbank.

Wenn Sie mit einem Recordset arbeiten, müssen Sie beachten, dass Sie das Recordset mit dem korrekten Cursortyp und der korrekten Sperreinstellung öffnen. Mit den Defaulteinstellungen (`adOpenForwardOnly`*,* `adLockReadOnly`*) können Sie das Recordset nicht bearbeiten.*

Tabelle 10.5 zeigt die `Recordset`-Methoden, die beim Bearbeiten von Daten verwendet werden.

Eigenschaft / Methode	Bedeutung
`AddNew` [*Fields*, *Values*]	`AddNew` erzeugt einen neuen, leeren Datensatz, den Sie dann wie einen vorhandenen Datensatz beschreiben und mit `Update` in die Datenbank schreiben können. Über die optionalen Argumente können Sie die Feldnamen und Feldwerte beim Aufruf direkt übergeben, was ich allerdings nicht für praktikabel halte. Beschreiben Sie besser nach `AddNew` die Felder separat (über die `Fields`-Auflistung).

Eigenschaft / Methode	Bedeutung
`Update`[*Fields*, *Values*]	`Update` schreibt die geänderten Daten in die Datenbank. Ein Aufruf von Update bewirkt, dass ADO eine `UPDATE`- bzw. `INSERT-INTO`-Anweisung an die Datenbank sendet, mit der die Daten dann aktualisiert bzw. angefügt werden.
`CancelUpdate`	Mit dieser Methode können Sie die Aktualisierung eines Datensatzes abbrechen. Das ist z. B. immer dann notwendig, wenn beim Schreiben in ein Feld oder beim Aufruf von `Update` ein Laufzeitfehler aufgetreten ist.
`Delete` [*AffectRecords*]	`Delete` löscht den aktuellen Datensatz, wenn Sie im Argument *AffectRecords nichts oder* `adAffectCurrent` *angeben. Geben Sie hier* `adAffectGroup` an, werden alle Datensätze gelöscht, die der in der `Filter`-Eigenschaft des Recordsets angegebenen Bedingung entsprechen.

Tabelle 10.5: Die Eigenschaften und Methoden zum Bearbeiten von Daten im ADO-Recordset-Objekt

10.3.3 Suchen von Datensätzen

Das Suchen von Datensätzen ist in jeder Datenbankanwendung ein sehr wichtiger Vorgang. Natürlich können Sie auch mit ADO eine Suche programmieren. Prinzipiell haben Sie dazu zwei Möglichkeiten: In einem geöffneten Recordset können Sie über dessen `Find`- oder `Seek`-Methode nach einem Datensatz suchen, der ein bestimmtes Kriterium erfüllt. `Seek` ist zwar schnell, weil diese Methode im Index einer Tabelle sucht, wird aber leider nur von den wenigsten Providern unterstützt. `Find` ist sehr langsam und erlaubt nur die Angabe eines einzigen Suchkriteriums. Außerdem ist es mit diesen Methoden schwierig, eine Suche zu programmieren, bei der auch mehrere Datensätze im Suchergebnis erscheinen können. Diese Variante beschreibe ich deswegen in diesem Buch nicht. Wenn Sie wissen wollen, wie Sie mit `Find` arbeiten, lesen Sie den Artikel »Suchen mit Find und Seek«, den Sie auf der Website zum Buch finden.

Eine wesentlich bessere Möglichkeit zur Suche nach Daten haben Sie, indem Sie ein Recordset beim Öffnen einfach filtern. Dazu verwenden Sie die WHERE-Klausel in der SQL-Anweisung, mit der Sie nahezu beliebig viele Filterkriterien angeben können. Wollen Sie z. B. alle Artikel auslesen, die der Kategorie 1 angehören und deren Einzelpreis kleiner 50 ist, sieht der Quellcode dazu etwa so aus:

```
Dim rstArtikel
Set rstArtikel = con.Execute("SELECT * FROM Artikel " & _
   "WHERE [Kategorie-Nr] = 1 AND Einzelpreis < 50")
```

Im Prinzip ist das ganz einfach. Sie müssen lediglich auf das Format der Suchwerte achten. Werte von Textfeldern müssen in Apostrophen eingeschlossen werden, Zahlen werden in der englischen Schreibweise angegeben. Bei Datumswerten wird es schwierig, weil die verschiedenen Datenbanksysteme ganz unterschiedliche Formate verwenden. Access will ein Datum z. B. in der englischen Form, eingeschlossen in #, übergeben bekommen (z. B: #31.12.2099# für den 31.12.2099).

Wenn Sie in Textfeldern über den LIKE*-Operator mit Wildcards suchen wollen, können Sie in ADO nicht »*« (o bis unendlich beliebige Zeichen) und »?« (genau ein beliebiges Zeichen) verwenden, sondern müssen immer »%« und »_« einsetzen.*

Der folgende Quellcode ermittelt alle Artikel, deren Name mit »A« beginnt:

```
Set rstArtikel = con.Execute("SELECT * FROM Artikel " & _
   "WHERE Artikelname LIKE "A%")
```

Bei einem bereits erstellten Recordset können Sie zum Filtern auch die Filter*-Eigenschaft einsetzen. In diese Eigenschaft können Sie eine komplexe Filterbedingung eingeben. Nach dem Setzen der* Filter*-Eigenschaft wird das Recordset entsprechend gefiltert. ADO macht dabei allerdings nichts anderes, als das Recordset zu schließen und mit der neuen* WHERE*-Bedingung wieder zu öffnen. Sie können den Filter wieder entfernen, indem Sie die* Filter*-Eigenschaft auf einen Leerstring oder auf die Konstante* adFilterNone *setzen.*

10.3.4 Sortieren

Wenn Sie ein Recordset sortieren wollen, können Sie dies idealerweise gleich beim Öffnen über die `ORDER BY`-Klausel der SQL-Abfrage erledigen:

```
rst.Open "SELECT * FROM Artikel ORDER BY Artikelname", _
   con, adOpenKeyset, adLockPessimistic
```

Alternativ können Sie die Sortierung vor oder auch nach dem Öffnen in der `Sort`-Eigenschaft des Recordsets einstellen. Wenn Sie die Sortierung hier allerdings erst nach dem Öffnen einstellen, wird das Recordset in der Regel neu abgefragt.

Ob das Setzen der `Sort`*-Eigenschaft nach dem Öffnen des Recordsets funktioniert, hängt ganz vom Provider und den Cursor-Einstellungen ab. Beim Microsoft-Jet-Provider z. B. müssen Sie dazu einen clientseitigen Cursor verwenden. Um Probleme zu vermeiden, sollten Sie das Recordset bei einer Umsortierung besser selbst schließen und mit der neuen Sortierung (die Sie dann einfach in* `Sort` *einstellen können) neu öffnen.*

Teil III

Go ahead!

11 ASP.NET-Grundlagen

Wenn auf einem Webserver das .NET-Framework installiert ist (siehe Kapitel 1), können Sie für die Programmierung von Internetanwendungen neben ASP auch ASP.NET verwenden. ASP.NET vereinfacht die Internetprogrammierung enorm: Umfangreiche, serverseitig ausgeführte Steuerelemente werden z. B. im Programm verwendet wie ganz normale Windows-Steuerelemente und erzeugen im Hintergrund browserspezifischen HTML-Code. Da ASP.NET-Programme grundsätzlich immer kompiliert werden, ist die Performance dieser Programme um einiges besser als in ASP. In der Vielzahl der Klassen in der Klassenbibliothek des .NET-Framework (der »Base Class Library«) finden Sie fast immer eine Lösung für ein Programmier-Problem. Die Programmierung unter .NET bietet aber noch viel mehr Vorteile, die ich hier gar nicht aufzählen kann. Die für Internetanwendungen wesentlichen Vorteile finden Sie auf Seite 300.

Dieses Kapitel behandelt also die Grundlagen der Internetprogrammierung mit ASP.NET. Ich beschreibe dazu zuerst kurz, was .NET überhaupt ist. Dann folgt eine kurze Beschreibung der einzelnen Teile des .NET-Framework. Da ASP.NET in diesem Buch eher praxisorientiert beschrieben wird, gehe ich nicht zu tief auf das .NET-Framework ein. Die Beschreibung *aller* Möglichkeiten des .NET-Framework würde auch (mindestens) ein eigenes Buch füllen. Im Prinzip müssen Sie aber nur wissen, wie Sie mit ASP.NET umgehen. Im .NET-Framework eine Lösung für ein Problem zu finden ist – zumindest mit der Hilfe der ASP.NET-Internetseiten (Seite 288) oder der Suche in Google und bei Microsoft (siehe Kapitel 1) – nicht allzu problematisch.

Ich zeige in diesem Kapitel dann noch, wie Sie prinzipiell mit ASP.NET arbeiten und Ihre ersten Anwendungen entwickeln. Dabei kann ich leider nicht alle Aspekte behandeln. ASP.NET ist sehr komplex und bietet viele Möglichkeiten. Ich kann in diesem Buch nur die wichtigsten vorstellen. Wenn Sie tiefer in ASP.NET einsteigen wollen, empfehle ich das Buch »Professional ASP.NET« aus dem Wrox-Verlag.

Die Beispiele dieses Kapitels können Sie in einem einfachen (HTML-) Editor nachvollziehen. Wenn Sie dazu allerdings die Microsoft-Entwicklungsumgebung verwenden wollen, beachten Sie bitte, dass diese schon erweiterte Techniken wie Code-Behind-Klassen (Seite 326) einsetzt und deshalb für die anfänglichen Beispiele etwas schwieriger zu verwenden ist.

.NET ist durchgehend objektorientiert. Zum Verständnis der zugrunde liegenden Technologien benötigen Sie gute Kenntnisse der OOP. Da ich diese hier aber nicht vermitteln kann, setze ich voraus, dass Sie wissen, was eine Klasse ist, was Vererbung bedeutet und was mit Überladen und Überschreiben von Methoden gemeint ist. Falls Sie in diesem Bereich noch Nachholbedarf haben, empfehle ich den begleitenden Artikel »OOP-Grundlagen«, den Sie auf der Website zum Buch finden.

Ein echter Testserver für ASP.NET

Wie ich bereits in Kapitel 7 beschrieben habe, können Sie Ihre ASP und ASP.NET-Programme in der realen Welt über den Web-Hoster Brinkster (*www.brinkster.com*) testen. Über die »General Membership« erhalten Sie – kostenfrei – 30 MB Server-Platz. Brinkster unterstützt ASP 3 und ASP.NET. Der Datei-Upload geschieht über eine einfach zu bedienende Web-Oberfläche. Lesen Sie aber vorher die »ASP Rules«, die Sie auf der »File Manager«-Seite finden (die automatisch geöffnet wird, nachdem Sie sich eingeloggt haben).

Wo Sie weitere Informationen finden

Neben den Suchmöglichkeiten, die ich bereits in Kapitel 1 genannt habe (Google und Microsofts MSDN-Suche), können Sie noch einige sehr gute Websites nutzen, um weitere Informationen zu ASP.NET zu finden oder Probleme zu lösen.

- Auf der deutschsprachigen Seite *www.aspheute.com* finden Sie einige gute Artikel zu ASP, ASP.NET und dessen Umfeld (C#, VB.NET). Nutzen Sie die Suchmöglichkeit auf der Startseite.

- DEVX (*www.devx.com*) bietet eigentlich eine Menge an Informationen, nicht nur zu .NET. Leider sind diese nicht allzu einfach zu erreichen (DEVX erhält damit den Sonderpreis für miese Navigation). Sie können bei DEVX aber natürlich auch suchen. Interessant ist der Newsletter, den Sie auf der Homepage abonnieren können. Haben Sie diesen für einen oder mehrere Programmierbereiche abonniert, erhalten Sie dann in unregelmäßigen Abständen eine E-Mail mit Infos und direkten Links zu neuen Artikeln.
- Gute Informationen finden Sie auch im ASP-Bereich von GOT DOT NET (*www.gotdotnet.com/team/asp/*). Der Link ASP.NET QUICKSTART TUTORIAL (*samples.gotdotnet.com/quickstart/aspplus/*) führt Sie zu vielen interessanten Grundlagenartikeln.
- Die Website von DEVASP *www.devasp.net/net/* liefert ebenfalls sehr viele Artikel und auch Beispiele, nicht nur zu ASP.NET. Die Website ist vor allen Dingen sehr übersichtlich aufgebaut.
- Bei *www.visualstudiowire.com* können Sie einen Newsletter abonnieren, der Sie per Mail mit Informationen zu neuen Artikeln (von verschiedenen Quellen) versorgt. Über diesen Newsletter habe ich schon sehr viele interessante Artikel gelesen, die ich sonst wahrscheinlich nie gefunden hätte.

11.1 .NET und das .NET-Framework

11.1.1 Was ist .NET?

Bevor ich das für Programmierer wichtige .NET-Framework beschreibe, will ich kurz klären, was .NET eigentlich ist: .NET (gesprochen als »Dotnet«) ist eine neue Strategie von Microsoft zur Erzeugung und Verteilung von Software. Die Grundlage von .NET ist das .NET-Framework. Das .NET-Framework ist eine komplexe (aber einfach anzuwendende) Infrastruktur, in der Sie Anwendungen programmieren, ausführen und verteilen können. Zum .NET-Framework gehören neben Diensten zur Erzeugung und Verwendung »normaler« Windows-Anwendungen auch solche zur Programmierung und Verwendung von Internetanwendungen. Dazu zählen neben den Webformularen (Web Forms), die mit ASP.NET entwickelt werden, auch die neuen Webdienste (Web Services). Webformulare (die in diesem

Kapitel grundlegend besprochen werden) erleichtern die Programmierung von Internetanwendungen. Webdienste sind im Prinzip Funktionen, die über das Internet aufgerufen werden können. Ein Webdienst, der auf dem Webserver einer Bank läuft, kann z. B. den aktuellen Kurs von Aktien über das Internet verfügbar machen.

Ein wichtiger Teil des .NET-Framework ist die Basis-Klassenbibliothek (BCL = Base Class Library). Diese Bibliothek enthält sehr viele Klassen, über die die meisten grundlegenden Programmierprobleme gelöst werden können. Wollen Sie z. B. in Ihrem ASP.NET-Programm eine Mail versenden? Kein Problem. In der BCL finden Sie dafür eine passende Klasse (`SmtpMail`). Bei der Anwendung dieser Klassen hilft dann ein wenig die Online-Hilfe zum Framework, aber meist viel besser eine der auf Seite 288 genannten Websites.

Die Basis-Klassenbibliothek kann ich in diesem Buch nur grundlegend erläutern. Die Vielfalt der Klassen ist einfach viel zu groß. Der Umgang mit der BCL ist aber meist sehr einfach (manchmal auch nicht so ganz einfach, aber da hilft das Internet weiter), sodass Sie ein Problem meist recht schnell selbst oder mit Hilfe des Internet lösen können.

Neben dem für Programmierer wichtigen Framework steht .NET aber auch für eine Vision. Basierend auf den neuen Webdiensten und den damit verbundenen Technologien verfolgt Microsoft mit .NET eine Idee, die als »Software as a Service« bezeichnet wird. Anwender sollen in Zukunft ihre Programme nicht mehr lokal auf dem Rechner speichern, sondern mehr oder weniger aus einzelnen Webdiensten und/oder den Funktionen von (Microsoft-)Produkten zusammenstellen. Die neuen Versionen der einzelnen Microsoft-Produkte werden deshalb durchgehend mit .NET-Support ausgestattet. Für Internetprogrammierer ist diese Vision unter Umständen ein wichtiges Thema: Falls sie sich durchsetzt, werden viele Firmen auf ihren Webservern Webdienste anbieten (wollen). Und irgendjemand muss diese ja programmieren.

11.1.2 Das .NET-Framework

Mit dem .NET-Framework setzt Microsoft eine (für Microsoft) neue Basistechnologie für Windows- und Internetanwendungen ein. Diese an Java erinnernde Technologie unterscheidet sich recht stark von den

bisher verwendeten. C#, Visual Basic.NET und alle weiteren .NET-Sprachen setzen auf diesem Framework auf und verwenden damit (normalerweise[1]) dieselben Grundkonstrukte (Datentypen, Bibliotheksfunktionen, das interne Verfahren zum Erzeugen und Verwenden von Klassen etc.). Sie sollten also wissen, was dieses Framework ist, bevor Sie sich mit einer .NET-Sprache beschäftigen. Bevor ich das .NET-Framework beschreibe, will ich aber kurz dessen Grundidee klären: Die Verwendung von Zwischencode für Programme.

Zwischencode-Programme und Just-In-Time-Compiler

Ab .NET basieren fast alle Microsoft-Programmiersysteme (»fast alle«, weil C++ eine Ausnahme bilden kann) nicht mehr auf dem Windows-API, sondern bestehen aus einem speziellen Microsoft-Zwischencode, der auch als *Intermediate Language* (IL) oder *Managed Code* bezeichnet wird. Dieser Zwischencode kann nicht direkt vom Betriebssystem ausgeführt werden, weil das Betriebssystem mit den speziellen Befehlen gar nichts anfangen kann. Um den Code ausführbar zu machen, wird dieser beim Aufruf von einem *Just-In-Time-Compiler* (JIT) in Betriebssystemcode umgewandelt. Dieser kompiliert dabei immer nur die Funktion, die gerade angesprochen wurde (deswegen heißt es auch »Just in Time«), behält den kompilierten Code für den nächsten Aufruf dann aber in seinem Cache. Ein Geschwindigkeitsnachteil ist nur für die ersten Aufrufe von Programmteilen zu erwarten, danach sollte ein IL-Programm genauso schnell ablaufen wie ein Programm, das das Betriebssystem direkt anspricht. Über ein spezielles Tool des .NET-Framework, dem »Native Code Generator« (NGEN.EXE), können Sie IL-Programme aber auch dauerhaft in nativen Maschinencode umwandeln und somit die Ausführung dieses Codes noch einmal beschleunigen.

Die Verwendung von Zwischencode für Anwendungen ergibt zwei große Vorteile: Wenn ein mit einer .NET-Programmiersprache geschriebenes Programm auf ein anderes Betriebssystem portiert werden soll, muss für dieses Betriebssystem »nur« das .NET-Framework inklusive passendem Just-In-Time-Compiler zur Verfügung stehen.

1. Einigen Sprachen wie C++ ist es möglich, Programme unter Umgehung des .NET-Framework zu erzeugen.

Das Programm muss weder neu geschrieben noch neu kompiliert werden. Microsoft hat angeblich bereits Rechte gesichert, das .NET-Framework auch auf (Corel-)Linux zu portieren. Obwohl viele Linux-Anhänger davon nicht gerade begeistert sein werden[2], ist diese Tatsache für uns »normale« Programmierer doch recht positiv. So können wir wahrscheinlich ohne viele Probleme ein Programm auf einem Windows-Rechner entwickeln und auf einem Linux-Rechner installieren und ausführen.

Der andere wichtige Vorteil ist, dass die Common Language Runtime (CLR), die den Zwischencode ausführt, die volle Kontrolle über die Ausführung des Codes besitzt. Sie kann deshalb u. a. überprüfen, ob die Ausführung des Codes sicher ist. Abstürze, die durch fehlerhafte Anwendungen verursacht werden, gehören damit wohl der Vergangenheit an.

Das .NET-Framework

Das .NET-Framework bietet .NET-Programmen sehr viele Dienste. Es besteht aus mehreren Basisklassen, der *Common Language Runtime* (CLR), einem oder mehreren Just-In-Time-Compilern und Compilern für C++-, C#-, Visual Basic.NET- und JScript-Quellcode. Abbildung 11.1 zeigt die Klassen und die Basisdienste des .NET-Framework.

Das Betriebssystem liefert natürlich die grundlegenden Dienste für alle Programme (bei Windows in Form des Windows-API). Diese Systemdienste werden allerdings von der Common Language Runtime (CLR) gekapselt.

Die Klassen des .NET-Framework sind sehr umfangreich. Allein die Kurzübersicht der einzelnen Namensräume (ein Namensraum organisiert einzelne Klassen thematisch) umfasst in der Online-Hilfe sieben Seiten. Und jeder Namensraum enthält mehrere, teils auch sehr viele Klassen. Im ersten Moment erscheinen diese Basisklassen sehr unübersichtlich. Aber die Organisation in Namensräumen erleichtert das Auffinden gesuchter Funktionalität enorm. Wenn Sie z. B. in Ihrer

2. Weil damit die Freiheit von Linux gefährdet wird. Kein Programmierer, der auf .NET basierende Programme entwickelt, kann dann noch sicher sein, mit seinen Programmen keine Rechte Dritter zu verletzen.

Anwendung eine Mail versenden wollen, suchen Sie im Namensraum `System.Web` (weil das Senden einer Mail ja etwas mit dem Web zu tun hat) und finden dort recht schnell den Namensraum `Mail` und dort die Klasse `SmtpMail`. Wenn Sie zur Programmentwicklung Visual Studio einsetzen, helfen Ihnen die IntelliSense-Features dieser IDE dabei, die passenden Klassen zu finden und zu verwenden. Der Inhalt eines Namensraums wird z. B. automatisch in einer Liste angezeigt, wenn Sie dessen Namen gefolgt von einem Punkt schreiben.

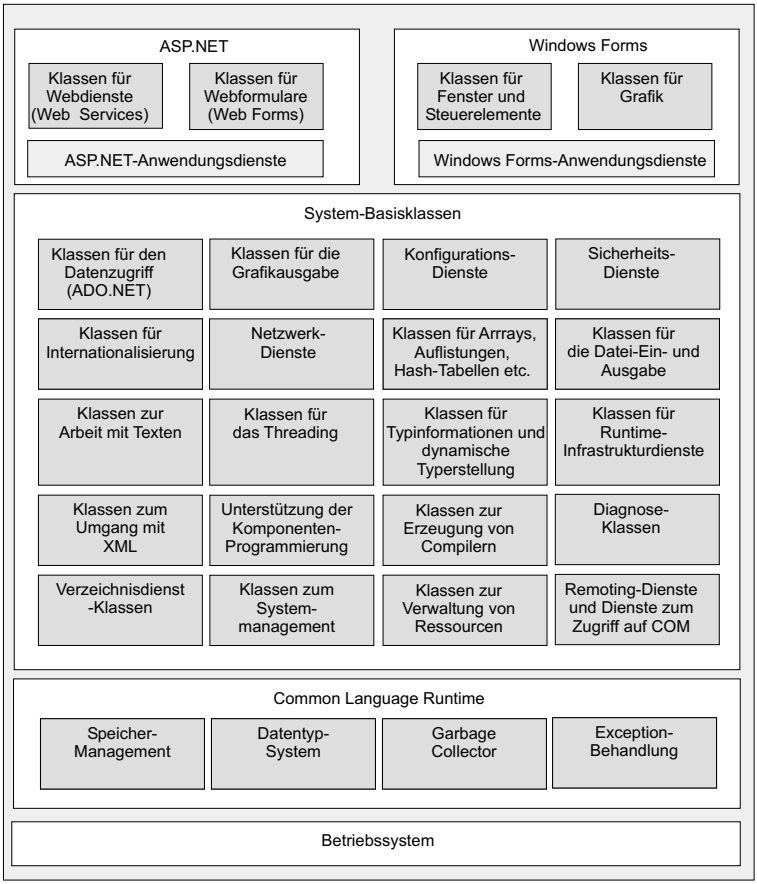


Bild 11.1: Das .NET-Framework (ohne Compiler)

Die Common Language Runtime

Die Common Language Runtime (CLR) ist eine Laufzeitumgebung für alle .NET-Programmiersprachen. Sie

- führt den IL-Code aus, der von einer beliebigen .NET-Programmiersprache erzeugt wurde. Dabei überprüft sie, ob der Programmcode für die Ausführung sicher ist. Die Überprüfung beinhaltet einen Test auf Typsicherheit. Ein versehentliches Überschreiben von fremden Speicherbereichen durch falsch verwendete Datentypen (z. B. der Versuch, einen Integer-Wert in einen Byte-Speicherbereich zu schreiben) ist unter der CLR nicht mehr möglich, weil diese unsicheren Programmcode erst gar nicht ausführt. Sie erzeugt stattdessen eine Ausnahme (Exception), die im Programm abgefangen werden kann,
- bietet eine große Anzahl an Standarddatentypen, die den Anforderungen moderner Programmiersprachen genügen,
- definiert die Regeln, die festlegen, wie neue Datentypen (Strukturen, Klassen) konstruiert werden müssen. Diese Regeln müssen von allen .NET-Programmiersprachen eingehalten werden, wenn diese CLR-kompatible Programme erzeugen sollen,
- übernimmt das Management von Objekten und deren automatische Zerstörung (über einen Garbage Collector[3]),
- übernimmt den Aufruf von Methoden in Objekten,
- und das Handling von Exceptions (Ausnahmefehlern).

Weil diese wichtigen Basisfeatures von allen .NET-Programmiersprachen verwendet werden, ist es mit .NET möglich, dass ein in einer Programmiersprache geschriebenes Programm (bzw. eine Komponente) ohne Probleme von einem in einer anderen Programmiersprache geschriebenen Programm verwendet wird. Die in der alten Welt bestehenden Probleme der Zusammenarbeit von Programmen, die in unterschiedlichen Sprachen geschrieben wurden (inkompatible Datentypen, unterschiedliches Handling von Exceptions etc.) gehören da-

3. Ein Garbage Collector (»Müllsammler«) ist ein Prozess, der immer dann, wenn das Programm gerade nicht beschäftigt ist, durch den Speicher des Programms geht und nicht mehr benötigte Objekte entfernt.

mit der Vergangenheit an. Eben diese Integration der verschiedenen Programmiersprachen ist ein wichtiger Bereich der CLR. Wenn Sie zum Beispiel in C# eine Komponente entwickeln, diese kompilieren und an andere Programmierer weitergeben, können diese die Klassen Ihrer Komponente z. B. in Visual Basic.NET ohne Probleme verwenden. Die Technik zur Verwendung von Objekten und zum Aufruf von Methoden und die verwendeten Datentypen sind eben identisch. Der VB-Programmierer kann sogar von Ihren Klassen neue Klassen ableiten (sofern Sie das nicht unterbinden). Löst Ihre Komponente bei bestimmten Ausnahmezuständen (oder im Fehlerfall) eine Ausnahme aus, kann der VB-Programmierer diese ganz einfach in einem Ausnahmeblock abfangen. Die unterschiedlichen .NET-Programme verstehen sich eben.

11.1.3 Namensräume, Assemblierungen und Module

Bei der Arbeit mit .NET begegnen Ihnen immer wieder die Begriffe »Typ«, »Namensraum« (Namespace), »Assemblierung« (Assembly) und »Modul«. Damit Sie wissen, worum es sich dabei handelt, beschreibe ich diese Begriffe kurz.

Typ

Jeder Datentyp wird in .NET als Typ (Type) bezeichnet. Dazu gehören Klassen, Strukturen, Aufzählungen (Enumerations), Schnittstellen und Arrays. Die normalen Datentypen wie `Integer`, `Double` und `String` sind in .NET übrigens grundsätzlich Klassen.

Namensräume

Das .NET-Framework fasst Typen immer in so genannten Namensräumen (Namespaces) zusammen. Ein Namensraum ist eine logische Gruppierung von Typen. Der Namensraum `System.Web` enthält z. B. alle Typen, die für die Arbeit mit dem Internet verwendet werden. Die Gruppierung vereinfacht die Suche nach Typen und erleichtert deren Zuordnung. Über Namensräume ist es sehr einfach, im .NET-Framework einen Typ zu finden, der eine bestimmte Funktionalität bietet. Sie müssen dazu nur, beginnend beim System-Namensraum, die einzelnen Namensräume durchgehen.

Namensräume werden normalerweise geschachtelt. Der `System`-Namensraum des .NET-Framework enthält einige Typen, aber auch wie-

der Namensräume. In `System` finden Sie die Namensräume `Data` (Klassen zur Arbeit mit Daten) und `Drawing` (Klassen zum Zeichnen von grafischen Elementen). Der `Data`-Namensraum enthält wiederum (u. a.) die Namensräume `OleDb` (die Klassen der zum Zugriff auf Datenbanken verwendeten ActiveX Data Objects, deren Basistechnologie OLEDB ist) und `SqlClient` (Klassen zum direkten Zugriff auf den SQL Server).

Die Dokumentation zum .NET-Framework stellt diese Namensräume sehr übersichtlich dar (Abbildung 11.2).

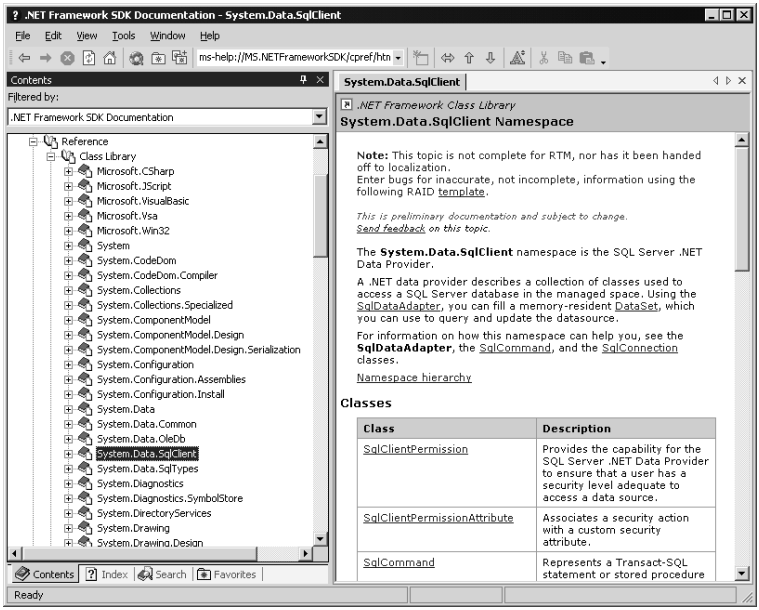


Bild 11.2: Die .NET-Framework-Dokumentation

Wenn Sie selbst Typen entwickeln, können Sie diese in eigenen Namensräumen anlegen und diese natürlich ebenfalls schachteln.

Assemblierungen und Module

Eine Assemblierung (englisch: »Assembly«) ist die Basiseinheit für die Verwaltung kompilierten Programmcodes und für die Verteilung von Anwendungen. Eine Assemblierung enthält den kompilierten Code einer

oder mehrerer Klassen und anderer Typen (Auflistungen, Strukturen etc.), alle Ressourcen (Bitmaps, Icons etc.), die von diesen Typen verwendet werden, und Metadaten, die die Assemblierung beschreiben.

Eine Anwendung ist unter .NET eine Kombination mehrerer Assemblierungen. Auch wenn Sie in Ihren Anwendungen selbst keine Assemblierungen einsetzen, verwenden Sie zumindest die Assemblierungen des .NET-Framework. Normalerweise enthalten einzelne Assemblierungen Typen und Ressourcen, die thematisch zueinander passen. Die Assemblierung SYSTEM.IO.DLL enthält z. B. Klassen und andere Typen für die Eingabe und Ausgabe von Daten.

Eine Assemblierung kann aus einer einzelnen kompilierten Datei, aber auch aus mehreren Dateien bestehen. Die einzelnen Dateien werden als Modul bezeichnet. Ein Modul enthält kompilierten Zwischencode (IL-Code), optional Ressourcen (z. B. Bitmaps und Icons) und Metadaten in Form einer .DLL- oder .EXE-Datei.

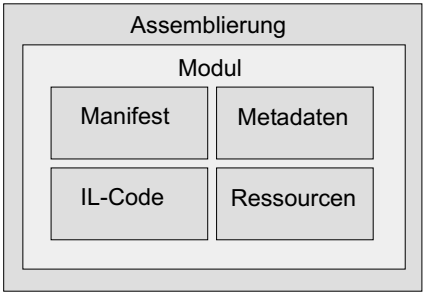


Bild 11.3: Eine Assemblierung, die nur ein Modul enthält

Metadaten beschreiben die im Modul enthaltenen Typen (also z. B. welche Klassen enthalten sind und wie die Eigenschaften, Methoden und Ereignisse der Klassen deklariert sind). Ein Modul der Assemblierung verwaltet normalerweise das so genannte Manifest (das aber auch in einer separaten Datei gespeichert sein kann). Das Manifest verwaltet die Abhängigkeiten der Assemblierung (inkl. der Version der verknüpften Ressourcen), verweist auf die enthaltenen Module, speichert den Namen des Autors, definiert eine Versionsnummer und beschreibt, welche (System-)Zugriffsrechte zur Ausführung der Assemblierung notwendig sind.

Damit beschreibt eine Assemblierung sich vollständig selbst. Es sind also keine weiteren Informationen notwendig, wie es z. B. im COM-Modell in Form von Registry-Eintragungen notwendig war. Deshalb kann eine Assemblierung einfach auf einen Computer kopiert und dort verwendet werden, was die Verteilung von .NET-Anwendungen im Vergleich zum COM-Modell erheblich vereinfacht.

Assemblierung werden oft aus mehreren Modulen zusammengesetzt. Das hat mehrere Gründe. Zum einen wird damit das Speichermanagement optimiert: Benötigt eine Anwendung nur bestimmte Typen einer Assemblierung, muss die CLR nur die jeweiligen Module in den Speicher laden und nicht die gesamte Assemblierung. Ein anderer Grund ist die Trennung von Programmcode und Ressourcen, um die Ressourcen z. B. für die Auslieferung in verschiedene Länder einfach austauschen zu können. Microsoft z. B. verwaltet viele Assemblierungen des .NET-Framework in zwei Modulen: Eines enthält den Programmcode, das andere die Ressourcen (wozu im Besonderen die Texte gehören, die im Programm verwendet werden). Abbildung 11.4 zeigt eine solche Assemblierung, die aus zwei Modulen besteht.

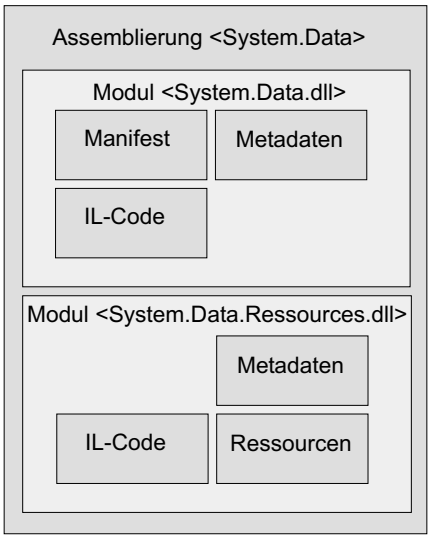


Bild 11.4: Eine aus mehreren Modulen zusammengesetzte Assemblierung am Beispiel von System.Data

Assemblierungen im Anwendungsordner und im Assemblierungen-Cache

Assemblierungen können einfach in dem Ordner gespeichert werden, in dem die Anwendung abgelegt ist, die diese Assemblierung verwendet. Unter ASP.NET sollten Sie diese Assemblierungen in einem dem Anwendungsordner untergeordneten Verzeichnis mit dem Namen BIN speichern. ASP.NET referenziert Assemblierungen im `Bin`-Ordner automatisch. Für selbst entwickelte Assemblierungen, die nur von einer Anwendung verwendet werden, wird die Speicherung im Anwendungsordner die Regel sein.

Jeder Computer, auf dem das .NET-Framework installiert ist, besitzt aber auch einen Ordner für Assemblierungen, der als Assemblierungen-Cache (Assembly Cache) bezeichnet wird. Diesen Cache finden Sie im Windows-Ordner unter dem Namen ASSEMBLY. Er speichert u. a. globale Assemblierungen, die von mehreren verschiedenen Anwendungen verwendet werden, und solche, die vor dem versehentlichen Löschen durch den Benutzer geschützt werden sollen. Einige Assemblierungen des .NET-Framework sind Bestandteil des globalen Teils des Assemblierungen-Cache.

Der Assemblierungen-Cache speichert daneben noch private, aus dem Internet heruntergeladene Assemblierungen (für Internet-Anwendungen) und den nativen Programmcode vorkompilierter Assemblierungen (die dann nicht mehr »just in Time« in nativen Code kompiliert werden müssen).

Assemblierungen und Namensräume

Normalerweise enthält eine Assemblierung alle Klassen eines (untergeordneten) Namensraums. Die Klassen des Namensraums `System.Net` sind z. B. in der Datei `System.Net.dll` gespeichert. Wenn Sie in Ihrem Quellcode mit den Klassen einer Assemblierung arbeiten wollen, müssen Sie auf dieses verweisen. ASP.NET-Programme sind aber bereits automatisch mit den wichtigen Assemblierungen verbunden.

11.2 Wichtige ASP.NET-Features

Wie Sie ja bereits gesehen haben, ist ASP.NET zunächst (einigermaßen) kompatibel zu ASP. Ein paar Änderungen im Quellcode, und schon laufen Ihre ASP-Programme auch unter ASP.NET. ASP.NET bietet aber einige weitere Features, die die Programmierung erst interessant machen.

Echte Programme

ASP.NET-Programme sind echte, kompilierte Programme. Wenn Sie eine ASPX-Seite zum ersten Mal aufrufen, wird diese automatisch in eine Assemblierung kompiliert. Bei folgenden Aufrufen verwendet das .NET-Framework dann die Assemblierung. Die Performance eines ASP.NET-Programms ist damit wesentlich höher als die eines ASP-Programms.

Bei der Programmierung nutzen Sie eine der verfügbaren .NET-Sprachen. Dabei handelt es sich meist um echte Programmiersprachen mit echten Datentypen, Typsicherheit, vollem Zugriff auf die BCL, strukturierter Fehlerbehandlung und mit der Möglichkeit, objektorientiert zu programmieren. Die Zeiten der Skriptsprachen in ASP und die damit verbundenen Probleme sind vorbei.

.NET-Komponenten

Viele Programmierer entwickeln für verschiedene Aufgaben spezielle Komponenten, die in mehreren Anwendungen eingesetzt werden können. In ASP.NET können Sie statt mit COM-Komponenten mit .NET-Komponenten arbeiten. Die Probleme, die COM-Komponenten mit sich bringen (Registration, Kompatibilitätsprobleme etc.) sind damit aufgehoben. .NET-Komponenten werden nicht registriert, beschreiben sich vollständig selbst und können ohne Probleme auch in mehreren Versionen auf einem Rechner gespeichert sein. Als besonderes Feature kann eine .NET-Komponente, die gerade noch in Benutzung ist, gegen eine neue Version ausgetauscht werden. Ein Herunterfahren des Servers ist nicht mehr notwendig.

Entwicklung ohne Entwicklungsumgebung

ASP.NET-Programme können komplett in einem einfachen Editor entwickelt werden. Wenn Sie für Ihre ASP.NET-Programme Komponenten entwickeln, können Sie diese über die .NET-Compiler VBC.EXE (Visual Basic) oder CSC.EXE (C#) kompilieren.

Debugging

Das .NET-Framework enthält einen integrierten Debugger, über den Sie Ihre Programme sehr flexibel entwanzen können.

Entwicklung in Visual Studio

Wenn Sie im Besitz von Visual Studio sind, können Sie die umfangreichen Features dieser Entwicklungsumgebung nutzen, um wesentlich produktiver zu arbeiten.

Einfache und flexible Konfiguration

Die Konfiguration einer Webanwendung erfolgt in ASP.NET über eine Datei WEB.CONFIG in XML-Form. Die Konfiguration kann deshalb sehr einfach per Hand oder mit Hilfe eines Programms geändert und angepasst werden. Jeder Unterordner kann eine eigene Konfigurationsdatei enthalten. Die Einstellungen übergeordneter Konfigurationsdateien werden automatisch in diese Datei vererbt. Die Konfiguration einer Webanwendung ist damit sehr flexibel. Ab Seite 331 finden Sie mehr dazu.

Umfangreiches Authentifizierungsmodell

ASP.NET integriert ein umfangreiches Authentifizierungsmodell. Neben dem normalen Windows-Modell können Sie mit recht wenig Aufwand eine eigene Benutzerverwaltung programmieren. Alternativ können Sie den so genannten Passport-Service nutzen, bei dem die Benutzer auf einem Microsoft-Server registriert sind. Leider bleibt in diesem Buch kein Platz für eine Beschreibung der Authentifizierung unter ASP.NET.

Serverseitige Steuerelemente

Neben den normalen HTML-Steuerelementen können Sie auf ASP.NET-Seiten auch spezielle serverseitige Steuerelemente einset-

zen. Auf der Seite werden diese über einen speziellen `asp`-Tag eingebunden. Diese Steuerelemente setzen sich selbst bei der Ausführung der Seite in reinen HTML-Code um. Dabei berücksichtigen Sie die Fähigkeiten des Browsers, der die Seite anfordert.

Das wirklich Interessante ist aber, dass Sie für diese Steuerelemente Ereignisprozeduren schreiben können, die serverseitig auf die jeweiligen Ereignisse der Steuerelemente reagieren. So können Sie innerhalb der ASP.NET-Seite zum Beispiel auf das `Click`-Ereignis eines Schalters reagieren, das generiert wird, wenn der Anwender den Schalter betätigt.

Validierung über Steuerelemente

Ein sehr hilfreiches ASP.NET-Feature sind die Validierungs-Steuerelemente. Diese Steuerelemente werden mit anderen Steuerelementen verbunden, deren Eingaben überprüft werden sollen. Spezielle Validierungs-Steuerelemente überprüfen, ob erforderliche Eingaben vorhanden sind, ob Eingaben in einem festgelegten Bereich liegen, oder verwenden einen regulären Ausdruck für die Überprüfung. Die Validierungs-Regeln geben Sie dazu einfach in den Eigenschaften der Steuerelemente an. Über ein benutzerdefiniertes Validierungs-Steuerelement können Sie in dessen `ServerValidate`-Ereignis eigene Validierungen vornehmen. Die speziellen Validierungs-Steuerelemente erzeugen wenn möglich JavaScript-Code, der die Validierung auf der Clientseite vornimmt. Server-Roundtrips sind dann nicht notwendig, was die Server-Ressourcen schont und die Anwendung beschleunigt. Die Validierungs-Steuerelemente werden in Kapitel 13 behandelt.

Datenbindung

Erinnern Sie sich noch daran, wie kompliziert es unter dem klassischen ASP war, Daten anzuzeigen, die Sie aus einer Datenquelle ausgelesen haben? Sie mussten Datensatz für Datensatz durchgehen und Feld für Feld ausgeben. In ASP.NET können Sie verschiedene Steuerelemente serverseitig einfach an eine Datenquelle anbinden. Die Anzeige der Daten übernimmt dann das Steuerelement selbstständig. Verschiedene Ereignisse dieser Steuerelemente helfen Ihnen, auch schwierigere Aufgaben, wie das Editieren der Daten, mit

recht wenig Programmcode zu lösen. Da die für Datenbindung wichtigen Steuerelemente sehr einfach und flexibel im Layout angepasst werden können, ist der Aufwand bei der Erzeugung von ASP.NET-Seiten für die Bearbeitung von Daten recht gering. In Kapitel 14 finden Sie mehr zu diesem Thema.

11.3 Wie werden ASP.NET-Seiten ausgeführt?

11.3.1 Das Framework und die automatische Kompilation

ASP.NET-Seiten werden nicht – wie bei ASP – von einer Script-Engine ausgeführt, sondern von Teilen des .NET-Framework. Die Dateiendung .aspx ist dazu im IIS mit der ISAPI-Erweiterung aspnet_isapi.dll verknüpft. Fordert ein Client eine Datei mit dieser Endung an, leitet der IIS die Anforderung zur aspnet_isapi.dll weiter. Diese gibt die Anforderung dann an das .NET-Framework weiter. Wird eine aspx-Datei zum ersten Mal angefordert, sorgt das .NET-Framework zunächst dafür, dass die Datei in eine Assemblierung kompiliert wird. Das Kompilieren ist ein recht aufwändiger Prozess, was Sie daran erkennen, dass die erste Anforderung einer .aspx-Seite recht viel Zeit in Anspruch nimmt. Die erzeugte Assemblierung wird dann in einem speziellen Systemcache unter einem eindeutigen Namen gespeichert. Sie finden diesen Cache auf dem Systemlaufwerk im Ordner \WINNT\Microsoft.NET\Framework\vx.x.xxxx\Temporary ASP.NET Files\. x.x.xxxx steht hierbei für die aktuelle Versionsnummer des Framework. Das Framework legt für jede ASP.NET-Anwendung einen separaten Ordner an und speichert dort (auf ziemlich komplizierte Weise) die Assemblierung und zugehörige Dateien. Die dort zu findende XML-Datei beschreibt z. B. die Abhängigkeiten der Assemblierung.

Wird eine bereits kompilierte aspx-Datei erneut angefordert, führt das .NET-Framework einfach die kompilierte Version aus[4]. Wird die Datei häufig angefordert, befindet sich die Assemblierung meist so-

4. Wie das Framework die Assemblierung lokalisiert, konnte ich nicht herausfinden. Der Name des Webordners spielt dabei eine Rolle. Die Namen der weiteren Unterordner des Cache bestehen aus Ziffern. Irgendwie ist der Name der aspx-Datei hier codiert oder wird an anderer Stelle mit der Assemblierung verbunden.

gar noch im RAM-Cache. Die Ausführung einer ASPX-Seite ist damit wesentlich schneller als die einer ASP-Seite (die ja immer wieder neu interpretiert wird).

Die Assemblierung enthält (wie alle Assemblierungen) Informationen über die Abhängigkeiten von anderen Dateien. Anders als bei normalen Assemblierungen werden hier aber auch Abhängigkeiten zu Quellcode-Dateien verwaltet. Die Assemblierung ist deshalb zumindest von der ASPX-Datei abhängig. Abhängigkeiten können aber auch zu anderen Dateien bestehen.

Vor der Ausführung der Assemblierung überprüft das Framework, ob eine der Dateien, von denen die Assemblierung abhängig ist, zwischenzeitlich geändert wurde (was wohl einfach durch einen Vergleich des Modifikationsdatums erfolgt). Wurde nur eine dieser Dateien verändert (bzw. besitzt eine der Dateien ein neueres Dateidatum), kompiliert das Framework die Assemblierung unter einem anderen Namen neu. Die alte Assemblierung wird mit der Dateiendung .DELETE gekennzeichnet und wohl irgendwann automatisch gelöscht (was auf meinem System allerdings bisher nicht der Fall war).

11.3.2 Die Page-Klasse: Eine ASP.NET-Seite ist ein Objekt

Wenn eine ASPX-Seite kompiliert wird, sorgt das .NET-Framework dafür, dass aus der einfachen Datei eine Klasse wird, die von der Klasse `System.Web.UI.Page` abgeleitet wird. ASPX-Seiten, die Code-Behind-Klassen verwenden (Seite 326), werden bereits im Quellcode von einer Klasse abgeleitet, die die `Page`-Klasse als Vorfahr hat. Einfache ASPX-Seiten ändert das Framework aber einfach so ab, dass diese von `System.Web.UI.Page` abgeleitet werden. Eine ASP.NET-Seite ist innerhalb der Assemblierung damit immer eine Klasse, die alle Eigenschaften, Methoden und Ereignisse der `Page`-Klasse besitzt. Die `Page`-Klasse beschreibe ich noch näher auf Seite 315. Zunächst reicht es aus, dass Sie wissen, dass diese Klasse alle für ASP.NET wichtigen Methoden und Eigenschaften beinhaltet. Das `Response`-, das `Request`- und die anderen ASP-Objekte sind z. B. Eigenschaften der `Page`-Klasse.

Wird die Assemblierung ausgeführt, erzeugt das Framework implizit ein Objekt dieser Klasse und führt dieses aus. Der Quellcode einer ASPX-Seite arbeitet deshalb eigentlich immer mit dem `Page`-Objekt.

Wenn Sie im Quellcode `Response.Write` verwenden, arbeiten Sie mit der `Response`-Eigenschaft der `Page`-Klasse. Das unterscheidet ASP.NET auch (u. a.) von ASP: In ASP wurden die spezifischen ASP-Objekte in der ASP.DLL instanziert und an die Script-Engine weitergegeben, bei ASP.NET sind diese Objekte einfach Bestandteil der `Page`-Klasse.

11.3.3 Das Rendering, die Ereignisbehandlung und der Viewstate

Bei der Ausführung erzeugt das `Page`-Objekt aus seinem Inhalt einen zum Client passenden HTML-Quellcode. Dieser Vorgang wird als »Rendering« bezeichnet. Die serverseitigen Steuerelemente der Seite werden dabei aufgefordert, ihren Quellcode selbst zu erzeugen. Der fertige HTML-Quellcode wird dann vom .NET-Framework zum Browser gesendet. Danach wird das `Page`-Objekt zerstört. Das `Page`-Objekt existiert also immer nur vom Beginn einer Anforderung bis zum Senden des HTML-Codes. Server-Resourcen werden nur dann reserviert, wenn diese auch konkret benötigt werden.

Beim Erzeugen des HTML-Codes werden serverseitige Buttons immer in Submit-Buttons umgewandelt, damit der Anwender das Formular absenden kann. In einem Formular ohne Buttons kann der Anwender seine Eingaben übrigens auch bestätigen, indem er einfach die Return-Taste betätigt. In einem automatisch hinzugefügten, versteckten Feld speichert die Seite den so genannten Viewstate (übersetzt etwa »Sichtstatus«). Dieser Status der Seite wird verschlüsselt gespeichert, sodass er im Internet nicht von anderen Personen gelesen werden kann.

Das folgende Beispiel soll zeigen, was aus einer ASPX-Seite wird, nachdem die Seite gerendert wurde. Die Seite reagiert auf die Betätigung eines Buttons dadurch, dass sie in einem Label eine Begrüßung ausgibt. Für den Text der Begrüßung verwendet die Seite den in einem Textfeld eingegebenen Namen des Benutzers:

```
<%@Page Language="VB"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
```

```
<title>Hello World in ASP.NET</title>

<script language="VB" runat="server">
 Sub HandleOK(sender As Object, e As System.EventArgs)
  If txtName.Text = "" Then
     lblHello.Text = "Hallo Unbekannter"
  Else
     lblHello.Text = "Hallo " & txtName.Text
  End If
 End Sub
</script>

</head>

<body>
<h1>Hello World in ASP.NET</h1>

<form id="frmInput" method="post" action="Hello_World.aspx"
runat="server">
 <p><asp:label id="lblHello" text="Hallo"
  runat="server"></asp:label></p>
 Name: <asp:textbox id="txtName" runat="server"></asp:textbox>
 <p>
 <asp:button id="btnOK" text="OK" width="100px"
  runat="server" onClick="HandleOK"></asp:button>
 </p>
</form>
</body>
</html>
```

Der erzeugte HTML-Quelltext sieht dann (wenn die Seite über den Internet Explorer aufgerufen wird) so aus:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">

<title>Hello World in ASP.NET</title>
```

```
</head>

<body>

<h1>Hello World in ASP.NET</h1>

<form name="frmInput" method="post"
 action="Hello_World.aspx" id="frmInput">
<input type="hidden" name="__VIEWSTATE"
 value="dDwxMjMyMjg0NjY1Ozs+" />

 <p><span id="lblHello">Hallo</span></p>
 Name: <input name="txtName" type="text" id="txtName" />
 <p>
 <input type="submit" name="btnOK" value="OK" id="btnOK"
 style="width:100px;" />
 </p>
</form>
</body>
</html>
```

Wie Sie sehen, wurde das ASP-Label in einen Span umgesetzt, die ASP-Textbox in ein `input`-Element und der ASP-Button in einen Submit-Button. Das `hidden`-Element mit dem Namen »__VIEWSTATE« (der wohl so definiert wurde, damit er eindeutig ist) speichert den Viewstate (was auch sonst ...).

Bestätigt der Anwender in der HTML-Seite seine Eingaben, fordert der Browser die ASP-NET-Seite erneut an (sie ist ja im `action`-Attribut des `form`-Tag angegeben). Das .NET-Framework erzeugt daraufhin ein neues `Page`-Objekt und führt dieses aus. Das `Page`-Objekt erkennt aber nun am gespeicherten Viewstate, dass es sich um einen Postback (einen Aufruf, der durch einen Formular-Submit ausgelöst wurde) handelt. Es vergleicht dann den im übertragenen `hidden`-Feld gespeicherten Viewstate mit den aktuellen Eingaben und führt für Steuerelemente, deren Status geändert wurde, gegebenenfalls die passende Ereignisprozedur aus. Hat der Anwender z. B. den Text einer Textbox geändert und für diese Textbox existiert eine `TextChanged`-

Ereignisprozedur, wird diese aufgerufen. Danach führt das `Page`-Objekt noch die `Click`-Ereignisprozedur des betätigten Button aus. Den betätigten Button erkennt das Objekt daran, dass der Browser für den betätigten Button dessen Beschriftung zurückgibt.

Die Ausführung der Ereignisprozeduren geschieht, bevor die Seite gerendert wird. Ändern Ereignisprozeduren den Status von Steuerelementen oder fügen diese der Seite neue Steuerelemente dynamisch hinzu, wirkt sich diese Änderung natürlich auf den danach erzeugten HTML-Quellcode aus. Mein Beispiel schreibt den Namen des Benutzers in das Label. Damit dieser neue Status der Seite auch bei den nächsten Aufrufen bestehen bleibt, wird er zusätzlich im Viewstate-Feld abgelegt. Nach der Eingabe des Namens »Zaphod« und der Bestätigung des Formulars sieht das Viewstate-Feld im erzeugten HTML-Dokument so aus:

```
<input type="hidden" name="__VIEWSTATE"
value="dDwxMjMyMjg0NjY1O3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDE+Oz47bDx0PHA8cDx
sPFRleHQ7PjtsPEhhbGxvIFphcGhvZDs+Pjs+Ozs+Oz4+Oz4+Oz4=" />
```

11.4 Die Sprache Visual Basic.NET (im Vergleich zu VBScript)

Ich möchte in diesem Buch nicht noch eine neue Sprache erläutern (JavaScript und VBScript wurden ja bereits kurz beschrieben). Die Beispiele des ASP.NET-Teils verfasse ich deswegen nicht in C# (obwohl ich das eigentlich lieber machen würde), sondern in Visual Basic.NET. Im Prinzip macht es aber keinen Unterschied, ob Sie Ihre ASP.NET-Seiten mit Visual Basic.NET, C# oder irgendeiner anderen .NET-Sprache entwickeln. Visual Basic.NET besitzt sehr viel Ähnlichkeit mit VBScript. Deshalb führe ich hier nur die Unterschiede zwischen diesen beiden Sprachen auf.

Eine ausführliche Beschreibung von Visual Basic.NET und C# finden Sie in den entsprechenden Büchern der Nitty-Gritty-Reihe. Dort werden auch die objektorientierte Programmierung unter .NET und die wichtigsten Klassen der Basis-Klassenbibliothek beschrieben.

Anweisungen

Anweisungen werden prinzipiell so geschrieben wie in VBScript. Wenn Sie Prozeduren oder Funktionen aufrufen, müssen Sie deren Argumente nun aber immer in Klammern übergeben. Das Schlüsselwort `Call`, das in VBScript noch verwendet wurde, um Argumente klammern zu können, fällt weg. Der Aufruf der `Write`-Methode des `Response`-Objekts sieht so aus:

```
Response.Write("Hello ASP.NET")
```

Datentypen und Variablen

Visual Basic.NET unterstützt die grundlegenden Datentypen der CLR. Bei der Deklaration von Variablen, Eigenschaften, Funktionsargumenten etc. sollten Sie davon Gebrauch machen. Sie geben den Datentyp in der Deklaration an:

```
Dim Variable As Datentyp
```

Wenn Sie mehrere Variablen desselben Datentyps deklarieren, müssen Sie den Datentyp nicht immer wiederholen. Sie können diesen lediglich für die letzte Variable angeben. Die ersten Variablen besitzen dann diesen Datentyp. Das folgende Beispiel deklariert drei Integer-Variablen:

```
Dim index1, index2, index3 As Integer
```

Wenn die Option `Strict` für die Seiten ausgeschaltet ist (was per Voreinstellung der Fall ist) und Sie bei einer Deklaration keinen Datentyp angeben, erhält die Variable den Datentyp `Object`:

```
Dim i ' i ist vom Typ Object
```

`Object` ersetzt in Visual Basic.NET (und C#) den Datentyp `Variant`, der nicht mehr existiert. In Visual Basic.NET sind alle Datentypen Objekte, die den Datentyp `Object` als gemeinsamen Vorfahr besitzen. Deswegen können Sie `Object` prinzipiell für alle Daten verwenden. Für Variablen vom Typ `Object` nimmt Visual Basic.NET ähnlich VBScript bei der Verwendung in Ausdrücken eine automatische Typkonvertierung vor. Sie sollten aber immer möglichst mit echten Datentypen arbeiten.

Um auch in Visual Basic Typsicherheit zu erhalten, sollten Sie die Option `Strict` *einschalten. Diese Option zwingt Sie dazu, Variablen, Eigenschaften, Argumente etc. mit Datentyp zu deklarieren, und schaltet die implizite Konvertierung von unterschiedlichen Datentypen aus. Per Voreinstellung ist diese Option ausgeschaltet. Dann können Sie im Quellcode z. B. einen String an eine Integer-Variable zuweisen, was zu schweren (logischen) Fehlern im Programm führen kann. Halten Sie es wie C#, schalten Sie die Option ein und konvertieren Sie explizit über die Cxyz-Funktionen von VB oder über die Methoden der* `System.Covert`*-Klasse (z. B.* `System.Convert.ToInt32` *für eine Konvertierung in einen Integer-Wert).*

Die Option `Strict` können Sie selbst für jede Seite in der @Page-Anweisung einschalten:

```
<%@Page Language="VB" Strict="true"%>
```

In Visual Studio können Sie diese Option auch in den Eigenschaften des Projekts so einstellen, dass diese automatisch für alle Seiten des Projekts gilt.

Visual Basic.NET kennt die in Tabelle 11.1 dargestellten Datentypen. Die Visual Basic.NET-Datentypen sind nur Synonyme für Basisdatentypen der CLR.

Datentyp	.NET-Datentyp	Wertebereich
`Boolean`	`System.Boolean`	Boolscher Wert (`True` oder `False`)
`Byte`	`System.Byte`	Ganzzahl von 0 bis 255 (ohne Vorzeichen)
`Char`	`System.Char`	beliebiges Unicode-Zeichen. Der Zahlwert dieses Datentyps geht von 0 bis 65535.
`Date`	`System.DateTime`	Datum vom 1. Januar 100 bis 31. Dezember 9999

Datentyp	.NET-Datentyp	Wertebereich
Decimal	System.Decimal	Fließkommazahl im Bereich +/- 79.228.162.514.264.337.593.543.950.335 ohne Dezimalzeichen und +/- 7,9228162514264337593543950335 mit 28 Dezimalstellen. Die kleinste Zahl ungleich Null ist +/- 0,0000000000000000000000000001.
Double	System.Double	Fließkommazahl von -1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte, 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte
Integer	System.Int32	Ganzzahl von –2.147.483.648 bis 2.147.483.647
Long	System.Int64	Ganzzahl von -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807
Object	System.Object	In einer Variablen vom Typ Object kann jeder beliebige Typ gespeichert werden.
Short	System.Int16	Ganzzahl von -32.768 bis 32.767
Single	System.Single	Fließkommazahl von $-3{,}402823 * 10^{38}$ bis $-1{,}401298 * 10^{-45}$ für negative Werte, $1{,}401298 * 10^{-45}$ bis $3{,}402823 * 10^{38}$ für positive Werte
String	System.String	Zeichenkette von 0 bis ca. 2 Mrd. Unicodezeichen

Tabelle 11.1: Die Datentypen von Visual Basic.NET

Die Funktionsbibliothek

In Visual Basic.NET können Sie mit den Funktionen arbeiten, die Sie in VBScript auch verwenden. Die spezifischen Visual Basic-Funktionen, die das .NET-Framework im Namensraum `Microsoft.VisualBasic`

zur Verfügung stellt, sind prinzipiell mit den VBScript-Varianten identisch. Da ASP.NET diesen Namensraum automatisch integriert, müssen Sie diesen beim Aufruf der Funktionen nicht angeben. Statt

```
Microsoft.VisualBasic.Replace(strNames, ";", "-")
```

können Sie also einfacher

```
Replace(strNames, ";", "-")
```

schreiben.

Neben diesen Funktionen können Sie aber natürlich auch direkt mit den Klassen des .NET-Framework arbeiten. Das hätte den Vorteil, dass Ihre Programme näher an Programmen liegen, die in C# oder anderen .NET-Sprachen geschrieben wurden. Ein Verstehen und Konvertieren solcher Programme wäre dann kein allzu großes Problem. Die spezifischen Visual Basic-Funktionen sind in Wirklichkeit nur Wrapper für Methoden der .NET-Basisklassen. Die `Replace`-Funktion finden Sie z. B. auch als Methode des `String`-Datentyps:

```
Dim strDemo As String, strResult As String
strDemo = "Zaphod;Trillian;Ford;Arthur"
strResult = strDemo.Replace(";", "-")
```


Alles ist ein Objekt

In .NET ist jeder Datentyp ein Objekt. Nicht nur komplexe, sondern auch einfache Datentypen wie `Integer` oder `Double` sind Objekte. Deshalb können Sie mit diesen Datentypen die Methoden aufrufen und Eigenschaften verwenden, die die zugehörige Klasse definiert. Das vorherige Beispiel nutzt schon die `Replace`-Methode der `String`-Klasse. Zahldatentypen wie `Integer` und `Double` besitzen z. B. die Methoden `ToString` (in einen String umwandeln) und `Parse` (aus einem String umwandeln) und die Eigenschaften `MaxValue` (größter speicherbarer Wert) und `MinValue` (kleinster speicherbarer Wert). Komplexere Datentypen wie z. B. `Date` besitzen wesentlich mehr Methoden und Eigenschaften. Insgesamt ist dieses System sehr konsistent. Sie finden bei den verschiedenen Datentypen immer wieder dieselben Methoden und Eigenschaften. Außerdem erleichtert dieses System die Arbeit mit den Daten: Wenn Sie auf einem Datentyp eine Operation ausführen wollen, müssen Sie eigentlich nur in dessen Methoden

und Eigenschaften nach einer Lösung forschen. Wirklich gut funktioniert dies allerdings nur in Visual Studio, denn diese Entwicklungsumgebung hilft Ihnen bei der Suche. Schreiben Sie den Punkt hinter einen Bezeichner, weil Sie eine Methode aufrufen oder eine Eigenschaft bearbeiten wollen, öffnen das IntelliSense-Feature von Visual Studio eine Liste mit den verfügbaren Elementen.

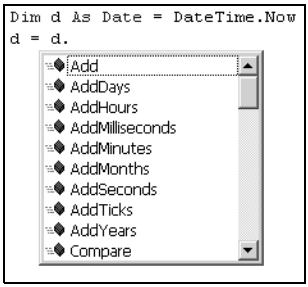


Bild 11.5: IntelliSense hilft beim Erforschen eines Objekts

Aus dieser Liste suchen Sie dann einfach das passende Element. Meist sagt der Name des Elements dabei schon alles. Wollen Sie dem Inhalt einer `Date`-Variablen eine bestimmte Anzahl Monate aufaddieren, finden Sie sehr schnell die Methode `AddMonths`:

```
Dim d As Date
d = d.AddMonths(3)
```

Methoden von Datentypen arbeiten grundsätzlich so, dass diese den ermittelten Wert zurückgeben. Damit wird eine maximale Flexibilität erreicht. Wenn Sie erreichen wollen, dass der ursprüngliche Wert des Datentyps geändert wird, müssen Sie die Rückgabe der Methode wieder dem Datentyp zuweisen (wie im Beispiel).

Fehlerbehandlung

Die Fehlerbehandlung funktioniert in ASP.NET-Seiten grundsätzlich anders als in VBScript. Ich habe diese bereits in Kapitel 7 beschrieben und gehe deshalb hier nicht mehr darauf ein.

Der Umgang mit Assemblierungen

Wie Sie ja bereits erfahren haben, sind die Klassen der Basis-Klassenbibliothek (BCL) des .NET-Framework in Namensräumen organisiert und in Assemblierungen gespeichert. Um eine Assemblierung verwenden zu können, muss diese zunächst im Programm referenziert werden. ASP.NET-Seiten referenzieren die wichtigen Assemblierungen wie SYSTEM.DLL, SYSTEM.WEB.DLL, SYSTEM.DATA.DLL u. a. automatisch. Wenn Sie aber eine Assemblierung verwenden wollen, die nicht automatisch referenziert wird, können Sie diese über die `@Assembly`-Anweisung einbinden:

```
<%@Assembly Name="Name"%>
<%@Assembly Scr="Dateiname"%>
```

Die erste Variante referenziert eine kompilierte Assemblierung (deren Name bekannt sein muss), die zweite Variante refenziert eine Assemblierung im Quellcode. In ASP.NET benötigen Sie diese explizite Referenzierung eigentlich nie: Die wichtigen Assemblierungen des .NET-Framework sind bereits referenziert, spezielle Assemblierungen legen Sie einfach in einem Ordner mit dem Namen BIN im Ordner der Webanwendung ab. Diese Assemblierungen referenziert ASP.NET dann auch automatisch.

Der Umgang mit Namensräumen

Wenn Sie Typen aus einem Namensraum verwenden wollen, müssen Sie normalerweise den kompletten Namen des Namensraums angeben. Die Klasse `Hashtable` (zur Erzeugung von Auflistungen) ist z. B. dem Namensraum `System.Collections` zugeordnet:

```
Dim ht As System.Collections.Hashtable
ht = New System.Collections.Hashtable()
```

.NET erlaubt aber auch das so genannte »Importieren« von Namensräumen. Ist z. B. der Namensraum `System.Collections` importiert, müssen Sie diesen bei der Verwendung der enthaltenen Typen nicht mehr angeben:

```
Dim ht As Hashtable
ht = New Hashtable()
```

Das spart natürlich enorm viel Schreibarbeit. In Sonderfällen kann dieses Vorgehen zu Problemen führen, nämlich dann, wenn zwei importierte Namensräume gleichnamige Typen beinhalten. Dann können Sie aber immer noch den kompletten oder vielleicht auch nur einen teilweisen Namen des Namensraums angeben.

ASP.NET ist mal wieder so nett, die (für ASP) wichtigen Namensräume implizit zu importieren: `System`, `System.Collections`, `System.Collections.Specialized`, `System.Configuration`, `System.IO`, `System.Text`, `System.Text.RegularExpressions`, `System.Web`, `System.Web.Security`, `System.Web.UI`, `System.Web.UI.WebControls`, `System.Web.HtmlControls`, `System.Web.Caching` und `System.Web.SessionState`.

Andere Namensräume, die nicht implizit importiert werden, können Sie über die `@Import`-Anweisung importieren:

```
<%@Import Namespace="Name"%>
```

11.5 Mit ASP.NET-Seiten arbeiten

11.5.1 Die Page-Klasse

Wie ich ja bereits auf Seite 304 erläutert habe, wird jede ASPX-Datei in eine `Page`-Klasse kompiliert, die bei der Ausführung der Seite automatisch instanziiert wird. Innerhalb der ASPX-Datei können Sie deswegen auf alle Eigenschaften und Methoden dieser Klasse zugreifen. Tabelle 11.2 beschreibt deshalb die wichtigsten Eigenschaften, Tabelle 11.3 die wichtigsten Methoden. Einige Eigenschaften besitzen große Ähnlichkeit mit ASP-Objekten. Die `Response`-Eigenschaft verweist z. B. auf ein `Response`-Objekt. Da ich diese Objekte bereits in Kapitel 8 beschrieben habe, verzichte ich hier auf eine nähere Erläuterung. Beachten Sie, dass viele der hier genannten Eigenschaften in der Konfigurationsdatei der Anwendung (Seite 331) für alle Seiten eines Ordners voreingestellt werden können.

Eigenschaft	Beschreibung
Application	verweist auf ein Application-Objekt
Cache	verweist auf ein Cache-Objekt. In diesem Objekt, das eine einfache Auflistung ist, können Sie Daten speichern, die zwischen notwendigen Server-Roundtrips (die oft von ASP.NET automatisch initiiert werden) bestehen bleiben. Die Daten werden dabei automatisch in hidden-Elementen auf den erzeugten HTML-Seiten gespeichert.
ClientTarget	Über diese Eigenschaft können Sie die automatische Erkennung des Browsertyps ausschalten und eigene Browsertypen angeben, deren Fähigkeiten die Seite voraussetzen soll.
EnableViewState	Hier legen Sie fest, ob die Seite den Status der Steuerelemente über hidden-Elemente auf den erzeugten HTML-Dateien verwaltet. Für Seiten, die keinen Viewstate benötigen, sollten Sie diesen ausschalten um die Performance zu erhöhen. Die Voreinstellung ist True.
ErrorPage	In dieser Eigenschaft können Sie eine URL angeben, die auf eine ASPX-Datei zeigt, die Sie für die Anzeige unbehandelter Laufzeitfehler verwenden wollen
IsPostBack	Diese Eigenschaft wird im Load-Ereignis der Klasse verwendet um festzustellen, ob die Seite aufgrund einer Client-Anforderung geladen wurde oder aufgrund eines von ASP.NET initiierten Server-Roundtrips
IsValid	IsValid wird gemeinsam mit speziellen Validierungs-Steuerelementen verwendet. Diese Steuerelemente überprüfen Eingaben des Benutzers auf Gültigkeit. IsValid gibt True zurück, wenn alle Validierungs-Steuerelemente Gültigkeit festgestellt haben.
Request	verweist auf ein Request-Objekt
Response	verweist auf ein Response-Objekt
Server	verweist auf ein Server-Objekt
Session	verweist auf ein Session-Objekt

Eigenschaft	Beschreibung
SmartNavigation	Über diese Eigenschaft können Sie die »smarte Navigation« für die Seite einschalten (siehe Seite 342).
Trace	Über diese Eigenschaft erhalten Sie Zugriff auf das TraceContext-Objekt der Seite. Wenn Tracing aktiviert ist (was über die Direktive @Page trace="true" geschehen muss), können Sie im Programmverlauf mit der Write-Methode Informationen in dieses Objekt schreiben, die dann zu Debuggingzwecken automatisch an das Ende der Seite angehängt werden.
TraceEnabled	Über diese schreibgeschützte Eigenschaft können Sie herausfinden, ob das Tracing eingeschaltet ist.
User	verweist auf ein Objekt der Klasse GenericPrincipal (normaler Benutzer) oder WindowsPrincipal (in Windows verwalteter Benutzer), über das Sie (in geschützten Webs) Informationen zum Benutzer abfragen können
Validators	eine Auflistung aller Validierungs-Steuerelemente der Seite

Tabelle 11.2: Die wichtigsten Eigenschaften der Page-Klasse

Methode	Beschreibung
DataBind()	DataBind wird verwendet, nachdem datengebundene Steuerelemente an eine Datenquelle angebunden wurden. Der Aufruf dieser Methode führt zur Anzeige der Daten.
FindControl (ByVal *Id* As String)	Über diese Methode können Sie ein Steuerelement auf der Seite über dessen ID suchen. Die Methode gibt einen Verweis auf das Steuerelement zurück.
LoadControl (ByVal *virtualPath* As String)	Über LoadControl können Sie ein externes ASP-Steuerelement aus einer .ASCX-Datei dynamisch laden. Die Methode gibt einen Verweis auf das Steuerelement zurück.

Methode	Beschreibung
`LoadTemplate (ByVal virtualPath As String)`	Über diese Methode können Sie eine Vorlage dynamisch laden.
`MapPath (ByVal virtualPath As String)`	`MapPath` mappt wie bei ASP einen virtuellen Pfad auf den physikalischen Pfad und gibt diesen zurück.
`Validate()`	Der Aufruf dieser Methode führt dazu, dass alle Validierungs-Steuerelemente der Seite die zu überprüfenden Eingaben validieren.

Tabelle 11.3: Die wichtigsten Methoden der Page-Klasse

Die `Page`-Klasse besitzt daneben auch noch einige Ereignisse, die Sie in der ASPX-Datei direkt auswerten können. Die wichtigsten zeigt Tabelle 11.4.

Ereignis	Beschreibung
`Init`	wird aufgerufen, wenn die Seite initialisiert wird
`Load`	wird aufgerufen, wenn die Seite geladen wird (nachdem alle Steuerelemente geladen wurden)
`Unload`	wird aufgerufen, wenn die Seite geschlossen wird, nachdem alle Informationen zum Browser gesendet wurden
`PreRender`	wird aufgerufen, kurz bevor die erzeugte HTML-Seite zum Browser gesendet wird
`AbortTransaction`	wird aufgerufen, wenn eine Transaktion, an der die Seite beteiligt ist, abgebrochen wird
`CommitTransaction`	wird aufgerufen, wenn eine Transaktion, an der die Seite beteiligt ist, bestätigt wird
`Error`	wird aufgerufen, wenn eine unbehandelte Ausnahme im Programm erzeugt wird. Dieses Ereignis können Sie für eine benutzerdefinierte Fehlerbehandlung verwenden.

Tabelle 11.4: Die wichtigsten Ereignisse der Page-Klasse

11.5.2 Klassisches ASP: Die Grundlage einer ASP.NET-Seite

Eine einfache (klassische) ASP.NET-Seite sieht im Quellcode zunächst fast so aus wie unter ASP. Die ASP-Anweisung `@Language` wird durch `@Page Language` ersetzt, als Sprache verwenden Sie eine der verfügbaren .NET-Sprachen. Innerhalb der ASP-Seite können Sie dann mit den normalen ASP-Token arbeiten. Das folgende Beispiel schreibt einen Begrüßungstext mit der aktuellen Zeit in einen Span:

```
<%@Page Language="VB"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
<title>Hello ASP.NET mit Inline-ASP</title>
</head>

<body>

<h1>Hello ASP.NET mit Inline-ASP</h1>

<span style="border:thin solid 2 black;
 height:22px;width:400px;background-color:royalblue">
<%
 Response.Write("Hallo, hier ist es gerade " & _
    DateTime.Now.ToShortTimeString() & " Uhr.")
%>
</span>

</body>
</html>
```

Wenn Sie innerhalb der Seite eigene Funktionen unterbringen wollen (was aber unter ASP.NET nicht sinnvoll ist, weil Sie dazu besser Code-Behind-Klassen und Komponenten einsetzen), können Sie diese nur in den `script`-Tag einbinden, nicht in die ASP-Token. Weiterhin sollten Sie beachten, dass Sie zum Datenzugriff vorzugsweise die neuen ADO.NET-Klassen verwenden (und nicht das alte ADO). Dasselbe gilt für alle weiteren externen Bibliotheken, die Sie unter ASP noch verwendet haben, wie z. B. die Scripting Runtime-Bibliothek zum Datei-

zugriff und zur Erzeugung von `Dictionary`-Objekten. Sie finden in den Klassen des .NET-Framework in der Regel einen vollwertigen und meist auch besseren Ersatz.

Ein wichtiger Unterschied zu ASP betrifft noch die Konfiguration einer Anwendung. Diese wird unter ASP.NET fast ausschließlich über eine XML-Datei (WEB.CONFIG, siehe Seite 331) eingerichtet und nicht mehr über die IIS-Administration.

Die @Page-Anweisung

Die `@Page`-Anweisung besitzt eine große Anzahl Attribute, über die Sie die Eigenschaften der Seite beeinflussen können. Die wichtigsten sind in Tabelle 11.5 aufgelistet.

Attribut	Bedeutung
`AspCompat`	Mit "true" stellen Sie die Seite so ein, dass diese in einem Single Threaded Apartment läuft und somit COM-Komponenten instanzieren kann, die ein solches Apartment erfordern. Die Voreinstellung ist "false".
`CodePage`	In diesem Attribut können Sie die Codepage definieren, die für den Response verwendet werden soll. Die Voreinstellung ist die Codepage des Servers.
`Debug`	Mit diesem Attribut ermöglichen Sie das Debugging der Seite. Die Voreinstellung ist "false".
`Enable Sessionstate`	Hier können Sie die Verwaltung des Sitzungsstatus für die Seite ausschalten um die Performance zu erhöhen. Die Voreinstellung ist "true".
`ErrorPage`	In diesem Attribut können Sie die URL einer ASPX-Seite angeben, die unbehandelte Ausnahmen anzeigen soll.
`Explicit`	Über dieses für VB gültige Attribut können Sie den eigentlich voreingestellten Zwang zur expliziten Variablendeklaration ausschalten, um dann in Ihren Programmen schwere logische Fehler machen zu können ☺.
`Inherits`	Dieses Attribut wird verwendet, wenn Sie Code-Behind-Klassen verwenden (siehe Seite).

Attribut	Bedeutung
Language	Hier geben Sie einen Bezeichner für die zu verwendende Sprache ein. Meist sind unterschiedliche Bezeichner erlaubt. Für Visual Basic können Sie z. B. "VB", "Visual Basic" oder "VBScript" angeben.
LCID	In diesem Attribut geben Sie die ID des Landes an, für das die länderspezifischen Einstellungen bei Zahl- und Datumsformatierungen verwendet werden sollen. Die Voreinstellung ist die LCID des Servers.
Src	Hier geben Sie den Dateinamen der Code-Behind-Klasse an, wenn Sie eine solche verwenden.
Strict	Mit diesem Attribut können Sie festlegen, dass auch unter Visual Basic Datentypen nur passend aufeinander zugewiesen werden können. Wenn Sie hier "true" eingeben, können Sie z. B. keinen String mehr zu einer Integer-Variable zuweisen, sondern müssen diesen explizit konvertieren. Damit erhalten Sie auch in Visual Basic Typsicherheit. Die Voreinstellung ist (leider) "false".
Trace	Mit Trace = "true" können Sie das Tracing (Spurverfolgung) für die Seite aktivieren. Die Informationen, die Sie mit den Methode der System.Diagnostics.Trace-Klasse in das Trace-Log schreiben, werden dann unten auf der Seite automatisch ausgegeben. Tracing nutzen Sie zum Debuggen komplizierter Fehler oder zum Feststellen von Performance-Problemen im Programm. Die Voreinstellung ist "false".
Transaction	Mit diesem Attribut können Sie das Verhalten der Seite in einer Transaktion steuern.

Tabelle 11.5: Die wichtigsten Attribute der @Page-Anweisung

11.5.3 Ereignisprozeduren: Trennung von Code und Inhalt

ASP.NET-Seiten arbeiten normalerweise nicht mehr mit ASP-Token, sondern trennen den Programmcode vom Inhalt der Seite. Auf einer einfachen Seite deklarieren Sie dazu eine oder mehrere Ereignisprozeduren in einem script-Tag. Diese Prozeduren reagieren auf die Ereignisse der Seite oder der Steuerelemente und können alle

serverseitigen Steuerelemente ansprechen. Der `script`-Tag wird mit dem Attribut `language` und `runat` definiert:

```
<script language="VB" runat="server">
 ' Ereignis- und andere Prozeduren
</script>
```

Vergessen Sie nicht das Attribut `runat="server"`. *Geben Sie dieses Attribut nicht an, werden Ihre Ereignisprozeduren einfach nicht ausgeführt.*

Eine Ereignisprozedur besitzt unter .NET immer dieselbe Struktur:

```
Private Sub Name( _
   ByVal sender As System.Object, _
   ByVal e As System.EventArgs) _
   [Handles Objektname.Ereignisname]
```

Verbindung einer Ereignisprozedur mit dem Ereignis

Ereignisprozeduren müssen mit dem Objekt und dem Ereignis verbunden werden, für das sie aufgerufen werden sollen. ASP.NET stellt Ihnen dazu gleich drei Möglichkeiten zur Verfügung.

Wenn das Attribut `AutoEventWireup` der `@Page`-Anweisung auf `True` eingestellt ist, können Sie eine Ereignisprozedur einfach über deren Namen mit einem Objekt und einem Ereignis verbinden. Die Prozedur muss dann folgendermaßen deklariert werden:

```
Private Sub Objektname_Ereignisname( _
   sender As System.Object, e As System.EventArgs)
```

Das folgende Beispiel zeigt den Kopf einer Prozedur, die das `Click`-Ereignis eines Buttons mit dem Namen *btnOK* bearbeitet:

```
Private Sub btnOK_Click(sender As Object, _
   e As System.EventArgs)
```

Die Voreinstellung des `AutoEventWireup`-Attributs (die in der Datei MACHINE.CONFIG vorgenommen wird) ist normalerweise `True`. Sie können dieses Attribut jedoch auch in der Konfigurationsdatei des Webordners (WEB.CONFIG) oder in der `@Page`-Anweisung anpassen:

```
<%@Page Language="VB" AutoEventWireup="True" %>
```

Die zweite Möglichkeit, ein Ereignis mit einer Prozedur zu verbinden, ist, bei der Deklaration des Steuerelements anzugeben, welche Prozedur für das Ereignis zuständig ist. Die Steuerelemente besitzen dazu Eigenschaften, die den gleichen Namen tragen wie die Ereignisse, allerdings mit einem »On« als Präfix. Das folgende Beispiel deklariert einen ASP-Button und verbindet das `Click`-Ereignis dieses Buttons mit einer Ereignisprozedur, die den Namen »HandleOK« trägt:

```
<asp:button id="btnOK"
 Text="OK" Width="100px"
 OnClick="HandleOK"
 Runat="server"/><br>
```

Die Prozedur kann nun einen beliebigen Namen tragen. Im Beispiel habe ich diese ja bereits »HandleOK« genannt:

```
Private Sub HandleOK(sender As Object, _
   e As System.EventArgs)
```

Aus dieser Technik ergibt sich ein großer Vorteil: Sie können eine einzige Ereignisprozedur für verschiedene Steuerelemente verwenden. Da Sie über das Argument *sender* eine Referenz auf das auslösende Steuerelement erhalten, können Sie dieses identifizieren und dann individuell reagieren.

Die dritte Variante, Ereignisprozeduren mit Ereignissen zu verbinden ist, der Ereignisprozedur mitzuteilen, für welches Ereignis sie geschrieben wurde:

```
Private Sub Name(sender As System.Object, _
   e As System.EventArgs) Handles Objektname.Ereignisname
```

Diese Variante funktioniert jedoch nur in Code-Behind-Klassen. Sie wird von Visual Studio eingesetzt, wenn Sie Ereignisprozeduren automatisch über die Entwicklungsumgebung erzeugen.

Die Argumente einer Ereignisprozedur

Alle Ereignisprozeduren besitzen prinzipiell unter .NET die gleichen Argumente. Lediglich der Datentyp des zweiten Arguments ist bei speziellen Ereignissen angepasst, damit den Ereignisprozeduren weitere oder andere Argumente übergeben werden können.

Das erste Argument (*sender*) ist eine Referenz auf das Objekt, das dieses Ereignis gefeuert hat. Über diese Referenz können Sie auf das Objekt zugreifen. Die Verwendung dieses Arguments ist zum Beispiel notwendig, wenn Sie eine Ereignisprozedur für mehrere Objekte verwenden. Zum Zugriff müssen Sie das Argument casten[5]. Verwenden Sie dazu in Visual Basic die `CType`-Funktion. Das folgende Beispiel castet im `Load`-Ereignis der Seite das Argument *sender* in ein Objekt der Klasse `Page`:

```
Dim p As System.Web.UI.Page
p = CType(sender, System.Web.UI.Page)
```

Im zweiten Argument *e* der Ereignisprozedur übergibt das aufrufende Objekt weitere Argumente, die sich je nach Ereignis unterscheiden. In ASP.NET besitzt dieses Argument nur bei wenigen Ereignissen Bedeutung. In Windows-Anwendungen werden hier bei einem Mausereignis (Klick, Doppelklick etc.) z. B. die Koordinaten der Maus übergeben.

Die Ereignisse der Seite (`Load`, `Unload` etc.) können als einzige auch ohne die genannten Argumente deklariert werden.

Das Hello-ASP.NET-Beispiel (Seite 319) sieht, um einen Button zum Aktualisieren erweitert, mit Code/Inhaltstrennung dann so aus:

```
<%@Page Language="VB"%>

<html>
<head>
<meta http-equiv="Content-Type"
 content="text/html; charset=iso-8859-1">
<title>Hello ASP.NET mit Code/Inhaltstrennung</title>

<!-- Code-Bereich -->
<script language="VB" runat="server">
  ' Ereignisprozedur für das Load-Ereignis der Seite
  Private Sub Page_Load(ByVal sender As System.Object, _
     ByVal e As System.EventArgs)
```

5. Casten bedeutet, einen Datentyp als einen anderen Datentyp auszuwerten.

```
    ' Wenn die Seite geladen wird
    If IsPostBack = False Then
       ' Wenn die Seite nicht aufgrund eines Submits
       ' aufgerufen wurde: Label initialisieren
       lblInfo.Text = "Hallo, hier ist es gerade " & _
          System.DateTime.Now.ToShortTimeString() & " Uhr."
    End If
  End Sub

  ' Ereignisprozedur für das Click-Ereignis des Button
  Private Sub HandleButton( _
     ByVal sender As System.Object, _
     ByVal e As System.EventArgs)
    lblInfo.Text = "Hallo, jetzt ist es genau " & _
       System.DateTime.Now.ToShortTimeString() & " Uhr."
  End Sub
</script>

</head>

<!-- Inhaltsbereich -->
<body>
<h1>Hello ASP.NET mit Code/Inhalt-Trennung</h1>

<form Id="frmInput" Runat="server">
 <asp:label id="lblInfo"
  runat="server" Height="22px"
  Width="400px" BackColor="RoyalBlue"
  BorderColor="Black" BorderStyle="Solid"
  BorderWidth="1px">Label</asp:label>
 <p>
 <asp:button id="btnReload" text="Aktualisieren" onClick="HandleBut-
ton" runat="Server"></asp:button>
 </p>
</form>
</body>
</html>
```

Das Beispiel nutzt das `Load`-Ereignis der Seite, um das Label zu beschreiben. Damit nicht bei jedem Postback neu geschrieben wird, fragt das Programm die `IsPostBack`-Eigenschaft des `Page`-Objekts ab und schreibt nur, wenn die Seite direkt vom Anwender geöffnet wurde. Die Betätigung des Aktualisieren-Schalters führt zur Ausführung des Click-Ereignisses, womit das Label neu beschrieben wird.

Der Programmcode ist sauber vom Inhaltsbereich getrennt. Eine Umformatierung des Inhalts beeinflusst den Quellcode nicht und ist aufgrund der Trennung auch einfacher.

11.5.4 ASP-Seiten mit Code-Behind-Klassen

Die Trennung von Code und Inhalt können Sie noch perfektionieren, indem Sie Code-Behind-Klassen einsetzen. Eine Code-Behind-Klasse ist eine separate Klasse, die entweder in einer Quellcode-Datei oder in einer Assemblierung vorliegt. Diese Klasse wird von der Klasse `System.Web.UI.Page` abgeleitet und erbt damit alle Eigenschaften, Methoden und Ereignisse der `Page`-Klasse. Diese neue Klasse geben Sie dann in der ASPX-Seite als Basisklasse an. Da die ASPX-Seite alle Elemente der Code-Behind-Klasse erbt, können Sie den Code der Seite komplett in diese Klasse auslagern. So können Sie den Inhaltsbereich der Seite z. B. in die Hände eines HTML-Designers geben, der sich dann nicht mit dem Programmcode herumschlagen muss.

Das folgende Listing zeigt das Beispiel des vorherigen Abschnitts, diesmal mit einer Code-Behind-Klasse:

```
Public Class Hello_Class
    Inherits System.Web.UI.Page

  ' Public-Eigenschaften, die mit den
  ' Server-Steuerelementen korrespondieren
  Public lblInfo As System.Web.UI.WebControls.Label
  Public btnReload As System.Web.UI.WebControls.Button

  ' Deklaration der Ereignisprozeduren

  ' Ereignisprozedur für das Load-Ereignis der Seite
  Public Sub Page_Load( _
```

```
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs)
  ' Wenn die Seite geladen wird
  If IsPostBack = False Then
     ' Wenn die Seite nicht aufgrund eines Submits
     ' aufgerufen wurde: Label initialisieren
     lblInfo.Text = "Hallo, hier ist es gerade " & _
        System.DateTime.Now.ToShortTimeString() & " Uhr."
  End If
 End Sub

 ' Ereignisprozedur für das Click-Ereignis des Button
 Public Sub HandleButton( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs)
   lblInfo.Text = "Hallo, jetzt ist es genau " & _
      System.DateTime.Now.ToShortTimeString() & " Uhr."
 End Sub
End Class
```

Die ASPX-Seite sieht dann deutlich vereinfacht aus:

```
<%@Page Language="VB" inherits="Hello_Class"
 src="Hello_Class.vb"%>

<html>
<head>
<meta http-equiv="Content-Type"
 content="text/html; charset=iso-8859-1">
<title>Hello ASP.NET mit Code/Inhalt-Trennung</title>
</head>

<body>

<form id="frmInput" method="post" action="Hello.aspx" runat="server">
 <asp:label id="lblInfo"
  runat="server" Height="22px"
  Width="400px" BackColor="RoyalBlue"
  BorderColor="Black" BorderStyle="Solid"
```

```
  BorderWidth="1px">Label</asp:label>
 <p>
 <asp:button id="btnReload" text="Aktualisieren" onClick="HandleBut-
ton" runat="Server"></asp:button>
 </p>
</form>

</body>
</html>
```

Wie Sie dem Beispiel entnehmen können, benötigt die Klasse Public-Eigenschaften, die denselben Namen und Datentyp besitzen wie die serverseitigen Steuerelemente. Die Datentypen der serverseitigen Web-Steuerelemente finden Sie im Namensraum `System.Web.UI.WebControls`, die der HTML-Steuerelemente im Namensraum `System.Web.UI.HtmlControls`. Innerhalb der Klasse können Sie nun genauso auf die Ereignisse des `Page`-Objekts und der Steuerelemente reagieren, wie innerhalb der ASPX-Seite selbst. Die Ereignisprozeduren müssen aber nun öffentlich (`Public`) oder geschützt (`Protected`[6]) sein, damit die abgeleitete Klasse darauf zugreifen kann.

In der ASPX-Seite müssen Sie aber nun bekannt machen, dass diese nicht direkt von der `Page`-Klasse, sondern eben von Ihrer Klasse abgeleitet wird. Dies geschieht durch die Attribute `inherits` und `scr` der `@Page`-Anweisung. In `inherits` geben Sie den Klassennamen an, in `src` die Quellcode-Datei oder Assemblierung, in der die Klasse gespeichert ist. Da die ASPX-Seite von der Code-Behind-Klasse abgeleitet wird, erbt diese alle Eigenschaften und Methoden der Klasse. Das vom .NET-Framework bei der Ausführung der Seite erzeugte Objekt kann damit auch die geerbten Ereignisprozeduren (die ja auch nur Methoden sind) ausführen.

11.5.5 Andere ASP.NET-Seiten öffnen

Die Beispiele der vorherigen Abschnitte öffnen immer wieder dieselbe ASP.NET-Seite, wenn der Anwender das Formular bestätigt. Ge-

6. Ein geschütztes Element einer Klasse verhält sich nach außen wie ein privates Element, kann aber im Gegensatz zu privaten Elementen in abgeleiteten Klassen verwendet werden. Aber das gehört zu den Grundlagen der OOP.

nau wie bei ASP müssen Sie aber häufig einfach andere Seiten öffnen. Meine erste Idee, die andere Seite einfach im Attribut `action` des `form`-Tag anzugeben, war nicht erfolgreich: ASP.NET überschreibt das `action`-Attribut eines serverseitig ausgeführten Formulars mit dem Namen der Seite, in der das Formular definiert ist. Das ist auch logisch, denn die Ereignisprozeduren müssen ja ausgeführt werden.

Die Lösung des Problems ist die, dass Sie in der Ereignisprozedur des Steuerelements, das die »Umleitung« bewirken soll, eine ganz klassische Umleitung programmieren. Die Werte, die Sie in der anderen Seite auswerten wollen, müssen Sie dann – auch wieder ganz klassisch – entweder über `Session`-Variablen oder über URL-Argumente übergeben.

Auf der folgenden Seite soll der Anwender seinen Namen eingeben und dann das Formular bestätigen:

```
<%@Page Language="VB"%>
<html>

<script language="VB" runat="Server">
 Private Sub HandleOK(sender As Object, _
    e As System.EventArgs)
   Response.Redirect( _
      "Auswerten.aspx?Name=" & txtName.Text)
 End Sub
</script>

<body>

<h1>Andere ASP.NET-Seiten aufrufen</h1>

Das ist die Seite, in der die Eingaben erfolgen.

<form id="frm" Runat="server" action="Test2.aspx">
 Dein Name: <asp:Textbox id="txtName" Width="150"
  Height="22" Top="100" Left="20" runat="server"/>
 <p>
 <asp:Button Id="btnOK" Text="OK" Width="100"
  OnClick="HandleOK" Runat="Server"/>
```

```
 </p>
</form>

</body>
</html>
```

Die Ereignisprozedur für das `Click`-Ereignis des Button leitet den Request zur Auswertungsseite um und übergibt die Eingabe in einem URL-Argument. Die Auswertunggseite liest dieses Argument im `Load`-Ereignis aus und schreibt den Wert in ein Label:

```
<%@Page Language="VB"%>
<html>

<head>

<script language="VB" runat="Server">
 Private Sub Page_Load(sender As Object, _
    e As System.EventArgs)
   lblHello.Text = "Hallo " & Request.QueryString("Name")
 End Sub
</script>

</head>

<body>

<h1>Andere ASP.NET-Seiten aufrufen</h1>

Das ist die Seite, die die Eingaben auswertet.

<form Name="frm" Runat="Server">
 <asp:Label Id="lblHello" Runat="Server"/>
</form>

</body>
</html>
```

Wahrscheinlich gibt es auch noch einen anderen, vielleicht eleganteren Weg, dieses Problem zu lösen. Die Diskussionen in den Dotnet-Newsgroups deuten aber auf die hier vorgestellte Lösung. Falls Sie eine bessere Lösung finden, schreiben Sie mir eine Mail.

11.6 Konfiguration einer ASP.NET-Anwendung

Die meisten Einstellungen der IIS-Konfiguration gelten für ASP.NET-Anwendungen nicht (mehr). Die Konfiguration einer ASP.NET-Anwendung erfolgt in einer XML-Datei mit dem Namen WEB.CONFIG. Diese Konfigurationsdatei ist normalerweise im Stammordner der Anwendung gespeichert. Zusätzlich dazu können in allen Unterordnern der Anwendung separate Konfigurationsdateien verwaltet werden. Die Einstellungen einer Konfigurationsdatei gelten immer für alle ASPX-Dateien in dem Ordner, in dem die Datei gespeichert ist, und in allen Unterordnern.

ASP.NET vererbt die Einstellungen einer übergeordneten Konfigurationsdatei in alle untergeordneten. Die untergeordnete Datei kann allerdings auch die Werte geerbter Attribute überschreiben.

Neben der Konfiguration für einzelne Webanwendungen existiert auch eine Datei für den gesamten Computer. Diese Datei finden Sie unter dem Namen MACHINE.CONFIG im Ordner \WINNT\MICROSOFT.NET\FRAMEWORK\VX.X.XXXX\CONFIG\ (X.X.XXXX = aktuelle Versionsnummer des .NET-Framework). Die Einstellungen dieser Datei werden in alle Webanwendungen vererbt. Besitzt eine Webanwendung keine WEB.CONFIG oder überschreibt nur einzelne Einstellungen, gelten die Einstellungen der MACHINE.CONFIG.

Eine typische Anwendungs-Konfigurationsdatei sieht aus wie im folgenden Listing:

```
<?xml version="1.0" encoding="Windows-1252" ?>
<configuration>
  <system.web>

    <httpRuntime maxRequestLength="500" />
```

```
    <compilation defaultLanguage="vb" debug="true" />

    <customErrors mode="RemoteOnly" />

    <authentication mode="Windows" />

    <authorization>
      <allow users="*" />
    </authorization>

    <trace enabled="false" requestLimit="10"
     pageOutput="false" traceMode="SortByTime"
     localOnly="true" />

    <sessionState mode="InProc"
    <httpHandlers>
      <add verb="*" path="*.vb"
       type="System.Web.HttpNotFoundHandler,System.Web" />
      <add verb="*" path="*.cs"
       type="System.Web.HttpNotFoundHandler,System.Web" />
      <add verb="*" path="*.vbproj"
       type="System.Web.HttpNotFoundHandler,System.Web" />
      <add verb="*" path="*.csproj"
       type="System.Web.HttpNotFoundHandler,System.Web" />
      <add verb="*" path="*.webinfo"
       type="System.Web.HttpNotFoundHandler,System.Web" />
    </httpHandlers>

    <globalization requestEncoding="utf-8"
     responseEncoding="utf-8" />

  </system.web>
</configuration>
```


`<configuration>` ist das Root-Element der Datei (XML schreibt das Vorhandensein eines einzelnen Root-Elements vor). Alle weiteren Elemente sind Bestandteile des Root-Elements. Der Bereich von `<system.web>` bis `</system.web>` ist eine Sektionsgruppe, die alle Einstellungen einer Konfigurationssektion zusammenfasst. In der MA-

CHINE.CONFIG sind neben `system.web` noch andere Sektionsgruppen wie z. B. `system.net` definiert, die andere Einstellungen speichern. Innerhalb dieser Sektionsgruppe werden einzelne Einstellungen vorgenommen. Dies geschieht manchmal in der Kurzform:

```
<httpRuntime maxRequestLength="500" />
```

Das `httpRuntime`-Element wird hier direkt mit Attributen definiert und mit `/>` abgeschlossen (das ist eine gültige XML-Syntax).

Andere Elemente, wie `authorization`, werden nicht über Attribute definiert, sondern über weitere Elemente, die in den Tag eingeschlossen werden:

```
<authorization>
  <allow users="*" />
</authorization>
```

Die verfügbaren Einstellungen sind sehr vielfältig. Mir fehlt hier einfach der Platz, um alle möglichen Elemente der Konfigurationsdatei zu erläutern. Ich beschreibe deshalb nur die wichtigsten. Bei den speziellen ASP.NET-Themen dieses Buchs finden Sie immer wieder Hinweise auf entsprechende Konfigurations-Einstellungen.

Achten Sie bei der Bearbeitung der Datei darauf, dass Sie die korrekte Groß-/Kleinschreibung verwenden. Die Bezeichner der Konfiguration nutzen das so genannte »camelCasing«: Worte werden grundsätzlich kleingeschrieben. Bei zusammengesetzten Worten wird der erste Buchstabe großgeschrieben. ASP.NET wertet die Konfigurationsdatei unter Berücksichtigung der Groß-/Kleinschreibung aus. Die korrekte Schreibung ist deswegen sehr wichtig.

Wenn Sie die Konfigurationsdatei in einem einfachen Editor bearbeiten wollen, müssen Sie im `encoding`*-Attribut des* `xml`*-Tag den Wert "Windows-1252" angeben, damit die Seite die Standard-Windows-Zeichentabelle verwendet. Visual Studio.NET setzt standardmäßig "UTF-8" ein. Mit dieser Codierung können Sie den Text der Datei nur in einem UTF-8-fähigen Editor bearbeiten.*

Visual Studio.NET erzeugt in einer Webanwendung immer eine voreingestellte Konfigurationsdatei mit recht guten Erläuterungen der einzelnen Elemente.

Allgemeine Einstellungen

Im `httpRuntime`-Element können Sie einige allgemeine Einstellungen für die Anwendung vornehmen. Das Beispiel zeigt die Voreinstellungen der Elemente:

```
<httpRuntime
   maxRequestLength="4096"
   executionTimeout="90"
   useFullyQualifiedRedirectUrl="false"/>
```

`maxRequestLength` steuert die maximale Größe der Anforderung (in KB). Dieses Attribut benötigen Sie, wenn Sie Dateien uploaden (wie es der Artikel »Dateien uploaden« auf der Nitty-Gritty-Website zeigt). Beim Uploaden müssen Sie die maximale Größe des Request manchmal höher einstellen, als es per Voreinstellung der Fall ist. Die Voreinstellung (in MACHINE.CONFIG definiert) ist 4096.

`executionTimeout` steuert den Zeitrahmen, der für die Ausführung einer Seite möglich ist (in Sekunden). Für Anwendungen mit Seiten, die sehr viel Programmcode beinhalten oder die evtl. lange auf das Öffnen einer Datenbankverbindung warten, müssen Sie die maximale Ausführungszeit höher einstellen, damit ASP.NET die Ausführung nicht vorzeitig abbricht.

Wenn Sie im Attribut `useFullyQualifiedRedirectUrl` "true" eingeben, werden URLs in Weiterleitungen (Redirect) immer mit dem vollen Servernamen qualifiziert (z. B. wird aus der URL *»/Shop/Default.asp«* die URL *»http://Servername/Shop/Default.asp«*). Einige für ASP.NET mögliche Endgeräte wie Handys arbeiten nur korrekt mit voll qualifizierten URLs.

Konfiguration der Seiten

Das folgende Beispiel (das wieder die Voreinstellungen verwendet) zeigt die möglichen Einstellungen für die Konfiguration von ASP.NET-Seiten:

```
<pages
  buffer="true"
  enableSessionState="true"
  enableViewState="true"
  enableViewStateMac="false"
  autoEventWireup="true" />
```

Mit dem Attribut `buffer` können Sie festlegen, ob eine Seite beim Erzeugen erst gepuffert und dann gesendet wird, wenn die Erzeugung beendet ist, oder ob die Seite sukzessive an den Browser gesendet wird. Wenn Sie die Pufferung für große Seiten abschalten, simulieren Sie einen schnelleren Response für den Anwender.

Über das Attribut `enableSessionState` können Sie die Verwaltung des Sitzungsstatus für die Seiten, die von der Konfiguration betroffen sind, abschalten. Falls Sie keine Sitzungsdaten speichern wollen, sollten Sie dies machen, um die Performance der Seiten zu erhöhen.

Über `enableViewState` können Sie die Speicherung des Viewstate (Seite 305) abschalten. Falls die von der Konfiguration betroffenen Seiten keinen Viewstate benötigen, sollten Sie diesen abschalten, um die Performance zu erhöhen.

Wenn Sie `enableViewStateMAC` einschalten, verwaltet ASP.NET einen zusätzlichen »Message Authentication Code« (MAC) auf der Seite, über den die Integrität der übertragenen Daten überprüft werden kann. Weil ein Einschalten dieses Attributs die Performance erheblich verringern kann, sollten Sie dieses Attribut nur dann einschalten, wenn Sie annehmen, dass die Daten auf dem Weg zwischen Client und Server auch wirklich beschädigt werden können.

Das Attribut `autoEventWireup` definiert, ob ASP.NET Ereignisse automatisch an Ereignisprozeduren weiterreicht. Die Voreinstellung ("true") bewirkt, dass Entwickler zum Abfangen eines Ereignisses nur eine passende Ereignisprozedur (wie z. B. `Page_Load`) schreiben müssen. Unter .NET ist dies eigentlich nicht üblich, Ereignisprozeduren müssen normalerweise mit dem Zusatz `Handles` *Objektname.Ereignisname* gekennzeichnet werden, damit die Ereignisprozedur auch mit dem Ereignis verbunden ist. Visual Studio setzt `autoEventWireup` z. B. auf "false".

11.7 Smarte Navigation

Die smarte Navigation ist ein besonderes Feature von ASP.NET, das zurzeit nur vom Internet Explorer ab Version 5.5 unterstützt wird. Bei dieser Art der Navigation werden bei einem Postback nur die Teile der Seite aktualisiert, die auch wirklich aktualisiert werden müssen (was intern über einen `iframe`-Tag realisiert wird). Ohne smarte Navigation verhalten sich Web-Anwendungen etwas störrisch, wenn der Anwender das Formular bestätigt und die Seite daraufhin neu angezeigt wird:

- Die neu erzeugte Seite wird wieder vom Anfang der Seite an angezeigt, die letzte Scrollposition geht also verloren,
- der Eingabefokus wechselt wieder auf das erste Element in der Tabulatorreihenfolge,
- der Bildschirm wird komplett neu aufgebaut,
- der letzte Stand der Seite wird in der Verlaufliste des Browsers gespeichert, sodass der Anwender über die Zurück-Funktion zum vorherigen Stand zurückgehen kann.

Smarte Navigation bewirkt, dass diese Nachteile (größtenteils) aufgehoben werden. Die aktuelle Position der Seite im Browser bleibt nach einem Submit bestehen. Der Bildschirm flimmert weniger, weil nur der geänderte Teil der Seite aktualisiert wird. Schließlich wird auch der letzte Stand der Seite nicht in der Verlaufsliste des Browsers gespeichert. Eine ASP.NET-Seite verhält sich damit ein wenig mehr wie eine Windows-Anwendung. Leider wechselte der Eingabefokus in meinen Tests doch immer wieder auf das erste Eingabeelement in der Tabulatorreihenfolge.

Sie schalten die smarte Navigation für einzelne Seiten entweder über das `SmartNavigation`-Attribut der Seite ein:

```
<%@Page Language="VB" SmartNavigation="true"%>
```

Oder Sie verwenden dazu die `SmartNavigation`-Eigenschaft der `Page`-Klasse (z. B. im `Load`-Ereignis der Seite):

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
```

```
SmartNavigation = True
...
```

Alternativ können Sie die smarte Navigation für alle Seiten der Anwendung in der Konfigurationsdatei (WEB.CONFIG) einschalten:

```
<configuration>
   <system.web>
      <pages smartNavigation="true"/>
   </system.web>
<configuration>
```

Die smarte Navigation funktioniert zurzeit nur mit dem Internet Explorer ab Version 5.5. Einschalten können Sie diese Navigation trotzdem, auch wenn andere Browser verwendet werden sollen. Das `Page`-Objekt setzt einfach die normale Navigation ein, wenn der Browser die smarte Navigation nicht unterstützt.

11.8 Debuggen von ASP.NET-Anwendungen

Sicher werden Ihre Webanwendungen, genau wie meine, den einen oder anderen Fehler enthalten. Diese Fehler können Sie mit dem Debugger des .NET-Framework finden und beseitigen. Enthält Ihre Anwendung Laufzeitfehler, die nicht abgefangen werden, zeigt ASP.NET zunächst die in der Konfiguration der Anwendung angegebene Fehleranzeige-ASPX-Seite an. Um diese Fehler debuggen zu können, müssen Sie der `@Page`-Anweisung der Seite zunächst das Attribut `debug` mit dem Wert "true" hinzufügen:

```
<%@Page Language="VB" debug="true"%>
```

ASP.NET zeigt den Laufzeitfehler dann bereits sehr informativ an (Abbildung 11.6).

Wenn Sie Fehler von entfernten Rechnern aus ebenfalls in dieser Form angezeigt lassen wollen, müssen Sie die Konfigurationsdatei (WEB.CONFIG) des Ordners so abändern, dass die vordefinierte Fehlerseite auf entfernten Rechnern nicht angezeigt wird.

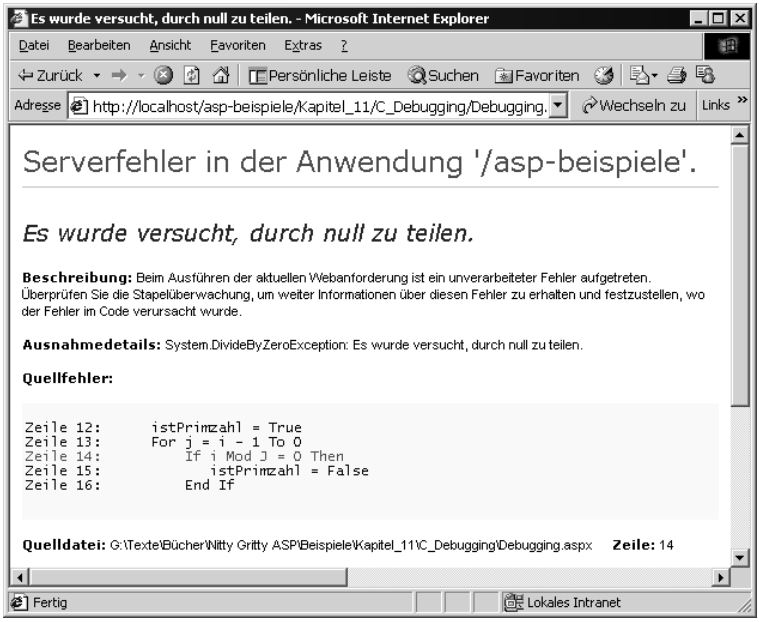


Bild 11.6: Anzeige eines Laufzeitfehlers im Browser, wenn das Debugging einer Seite eingeschaltet ist


```
<?xml version="1.0" encoding="Windows-1252" ?>
<configuration>
    <system.web>
        <customErrors mode="Off"/>
    </system.web>
</configuration>
```

Normalerweise ist der Wert von `customErrors` auf "RemoteOnly" eingestellt, damit das Debugging per Browser nur funktioniert, wenn der Browser auf dem Webserver selbst läuft.

Für Fehler, deren Ursache Sie so nicht finden, können Sie nun den Debugger bemühen. Wenn Sie Visual Studio besitzen, können Sie den dort integrierten Debugger einsetzen. Ansonsten verwenden Sie den Debugger der Common Language Runtime (CLR), der Bestandteil des .NET-Framework ist.

Der CLR-Debugger

Den Debugger der CLR finden Sie unter dem Namen DbgCLR.exe im Ordner \Programme\Microsoft.NET\FrameworkSDK\GuiDebug. Nachdem Sie ihn gestartet haben, verbinden Sie den Debugger mit dem Arbeitsprozess von ASP.NET. Öffnen Sie dazu den Dialog zur Verbindung mit den laufenden Prozessen (Extras | Debugprozesse). Schalten Sie die Option Systemprozesse anzeigen ein und markieren Sie den Prozess aspnet_wp.exe.

Klicken Sie dann auf Anfügen, um den Debugger mit diesem Prozess zu verbinden. Nun können Sie eine Seite in den Debugger laden (Datei | Öffnen oder einfach vom Explorer in den Debugger ziehen). Jetzt können Sie Haltepunkte setzen, durch den Programmcode gehen und Werte von Variablen oder Ausdrücken überwachen. Das Vorgehen unterscheidet sich dann nicht mehr vom Debuggen in Visual Studio, weswegen ich dies im nächsten Abschnitt beschreibe.

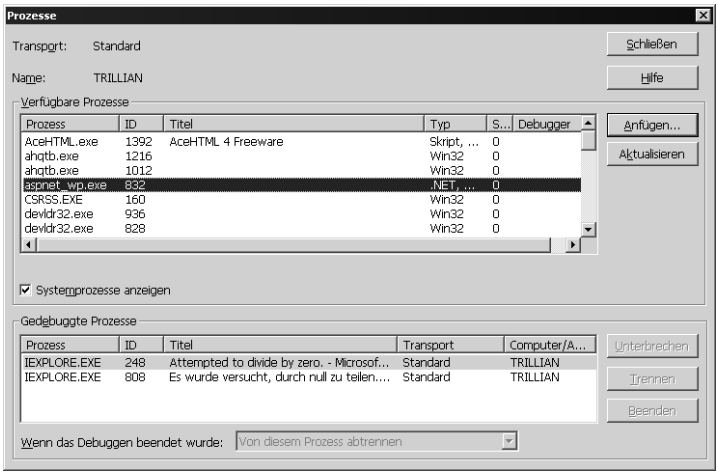


Bild 11.7: Verbindung mit dem ASP.NET-Arbeitsprozess im CLR-Debugger

Visual Studio

In Visual Studio ist das Debuggen etwas einfacher als mit dem CLR-Debugger, weil Sie Ihre Anwendung direkt mit F5 starten und damit automatisch mit dem Debugger verbinden können.

Visual Studio besitzt gegenüber dem CLR-Debugger einen großen Vorteil: Sie können den Programmcode während der Ausführung im Debugger mit einigen Ausnahmen auch verändern. Einschränkungen dabei sind:

- Innerhalb von Funktionen können Sie lediglich neue Variablen bis zur Gesamtgröße von maximal 64 Byte deklarieren,
- Ressourcen-Dateien können nicht geändert werden,
- Ausnahme-Behandlungs-Blöcke können nicht geändert werden,
- Sie können keine Datentypen ändern und
- Sie können keine Funktionen entfernen.

Das Debuggen

Wenn Sie innerhalb einer Programmdatei Ihrer Anwendung mit F9 an einer ausführbaren Anweisung einen Haltepunkt setzen, hält der Debugger das Programm dann an dieser Anweisung an und Sie können den Fehler suchen.

Mit F11 gehen Sie einen Schritt im Programm weiter. Der Debugger verzweigt dann in Methoden, Funktionen und Prozeduren, die in den zu testenden Anweisungen aufgerufen werden. Wollen Sie diese überspringen, betätigen Sie stattdessen F10.

Befindet sich das Programm im Haltemodus, können Sie im Befehlsfenster (Menü DEBUGGEN | FENSTER | DIREKT) beliebige Ausdrücke testen. Geben Sie dazu ein Fragezeichen gefolgt vom Ausdruck ein. Hilfreich sind auch die Fenster AUTO, LOKAL und AUFRUFLISTE, die nur angezeigt werden können, wenn das Programm sich im Haltemodus befindet. Im Auto-Fenster können Sie den Inhalt von Variablen oder Eigenschaften (automatisch) anzeigen lassen, während Sie mit F10 oder F11 durch das Programm gehen. Das Lokal-Fenster zeigt automatisch alle lokalen Variablen und Eigenschaften an (das sind die, die gerade in der aktuellen Prozedur gültig sind). Die Aufrufliste zeigt eine Liste der Prozeduraufrufe, die zum aktuellen Haltepunkt geführt haben. Hier können Sie beispielsweise zu einer übergeordneten Prozedur wechseln, wenn der Fehler durch ein fehlerhaft übergebenes Argument verursacht wurde.

Bedingte Haltepunkte

Für schwieriger zu findende Fehler können Sie bedingte Haltepunkte definieren. Diese speziellen Haltepunkte besitzen eine Bedingung, die wahr werden muss, damit der Debugger anhält. Außerdem können Sie noch festlegen, wie oft diese Bedingung erfüllt sein muss, damit der Haltepunkt aktiv wird. Solche Haltepunkte können Sie wie normale Haltepunkte mit Anweisungen verbinden. Setzen Sie dazu einen Haltepunkt und öffnen Sie den Eigenschaftendialog für den Haltepunkt (Rechtsklick auf dem Haltepunkt | HALTEPUNKTEIGENSCHAFTEN). Über den Schalter BEDINGUNG können Sie die Bedingung definieren und über den Schalter TREFFERANZAHL die Anzahl der Treffer einstellen. Eine Bedingung muss immer komplett und in der verwendeten Programmiersprache formuliert werden. Wenn Sie eine Variable i daraufhin überwachen, ob deren Wert 0 wird, können Sie nicht einfach »= 0« eingeben, sondern müssen komplett »i = 0« schreiben. Für das Beispiel mit der Divison durch 0 (Abbildung 11.6 auf Seite 338) wäre vielleicht ein bedingter Haltepunkt sinnvoll, der anhält, wenn die Variable j den Wert 0 erreicht.

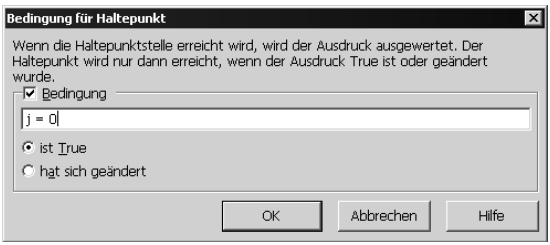


Bild 11.8: Der Dialog zur Einstellung der Bedingung(en) für einen Haltepunkt

Haltepunkte an Ausnahmen

Wenn Sie genauso programmieren wie ich (schnell, oft unüberlegt, meist ohne Vorbereitung), werden Ihre Anwendungen wahrscheinlich auch genauso viele Ausnahmen (Laufzeitfehler) erzeugen wie meine. Normale Haltepunkte helfen bei der Suche nach der Ursache häufig nicht weiter. Sie können den Debugger aber auch so einstellen, dass dieser bei allen oder nur bestimmten Ausnahmen anhält. Öffnen Sie dazu den Ausnahmen-Dialog (Menü DEBUGGEN | AUSNAHMEN). Im CLR-Debugger müssen Sie zunächst eine Verbindung zum

ASPNET_WP.EXE-Prozess aufbauen, damit Sie diesen Dialog zur Verfügung haben. Hier können Sie dann einstellen, bei welchen Ausnahmen der Debugger anhalten soll. Abbildung 11.9 zeigt, wie Sie den Debugger einstellen, damit dieser bei allen Ausnahmen anhält.

Die Einstellung Wenn die Ausnahme nicht behandelt wird ist für ASP.NET nicht relevant, da ASP.NET alle Ausnahmen implizit (durch die Anzeige einer Fehlerseite) behandelt. Unbehandelte Ausnahmen gibt es also normalerweise nicht (es sei denn, Sie schalten die implizite Fehlerbehandlung irgendwie ab).

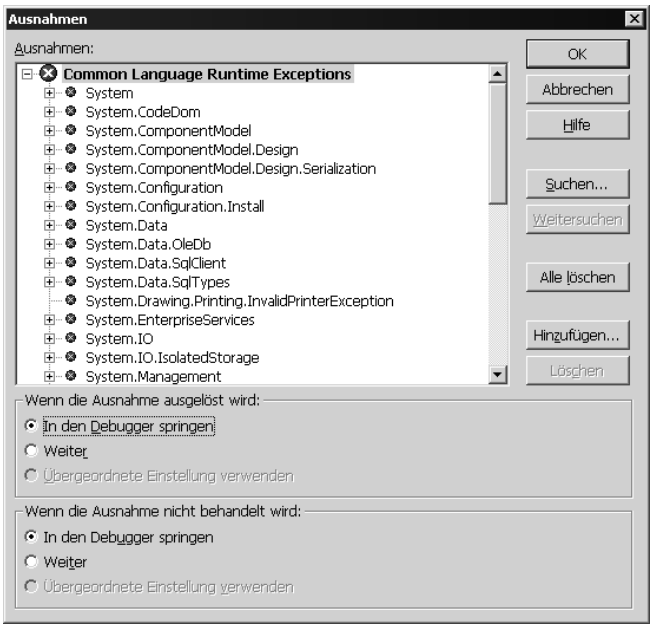


Bild 11.9: Der Dialog zur Einstellung des Verhaltens des Debuggers bei Ausnahmen

Um zu vermeiden, dass der Debugger nun bei allen (auch behandelten) Ausnahmen anhält, können Sie das Anhalten nur für den Typ der erzeugten Ausnahme einschalten. Den Typ lesen Sie einfach aus der ASP-NET-Fehlermeldung aus. Eine Division durch 0 erzeugt z. B. eine `DivideByZeroException` (siehe bei Abbildung 11.6 auf Seite 338). Damit wird das Debuggen von Ausnahmen wesentlich vereinfacht.

12 Visual Studio.NET

Visual Studio.NET, die umfangreiche Microsoft-Entwicklungsumgebung für .NET, eignet sich hervorragend zur Erstellung von ASP.NET-Anwendungen. Neben der Möglichkeit, Webanwendungen visuell zu erstellen, helfen besonders auch die IntelliSense-Features enorm dabei, Problemlösungen zu finden und produktiv zu sein. Visual Studio.NET ist nicht nur für die Erzeugung von Webanwendungen gedacht. Sie können damit auch normale Windowsanwendungen, Klassenbibliotheken, Windowsdienste und andere Programme erzeugen. Dieses Kapitel beschreibt aber lediglich die Verwendung von Visual Studio.NET zur Erzeugung von Webanwendungen. Ich erläutere dabei aber nicht jeden Menüpunkt und jede Funktion, sondern nur das grundsätzliche Vorgehen. Lassen Sie sich nicht von der Vielzahl der Funktionen dieser Entwicklungsumgebung verwirren. Gehen Sie einfach die Menüs in Ruhe durch und erforschen Sie die Möglichkeiten. Der Editor besitzt einige gute Features, wie z. B. das automatische Formatieren einer Quellcodedatei, die Sie recht einfach in den Menüs finden.

12.1 Der Start und das Erzeugen neuer Webanwendungs-Projekte

Erstellen eines neuen Webanwendungs-Projekts

Wenn Sie Visual Studio starten, können Sie im Startdialog neue Projekte erstellen oder vorhandene öffnen. Klicken Sie u. U. im linken Bereich auf den Link Beginnen, um den Startdialog anzuzeigen.

Ein Web-Projekt besteht aus den aspx-Dateien, Code-Behind-Klassen und anderen Komponenten, die Sie zu einer Webanwendung zusammensetzen. Visual Studio verwendet eine Projektdatei mit der Endung .sln (Solution), die alle Bestandteile und Einstellungen des Projekts verwaltet. Die Projektdatei wird automatisch im Ordner \Dokumente und Einstellungen\Benutzername\Eigene Dateien\Visual Studio-Projekte\ gespeichert. Die erzeugten aspx- und andere Quellcodedateien speichert Visual Studio im Ordner der Webanwendung.

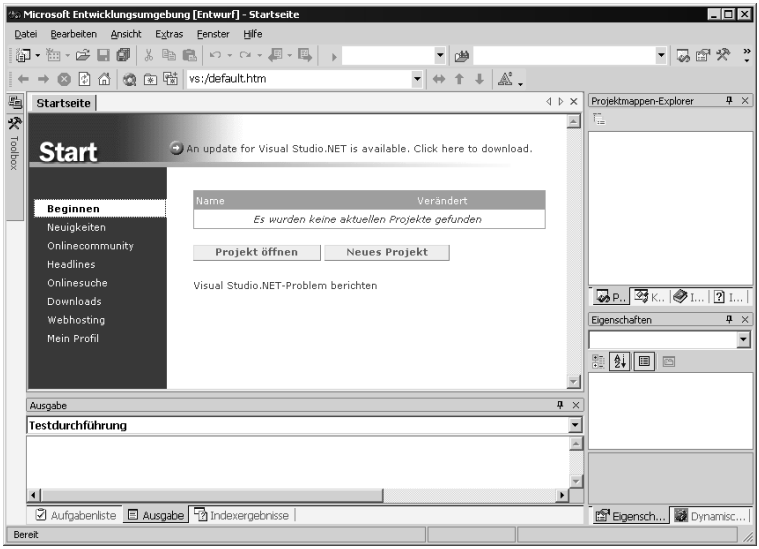


Bild 12.1: Visual Studio.NET

Klicken Sie auf den Schalter NEUES PROJEKT des Visual Studio-Startdialogs, um ein neues Projekt zu erstellen. Im dann folgenden Dialog können Sie unter den verfügbaren Möglichkeiten die zur Erzeugung von Webanwendungen auswählen (Abbildung 12.2).

In diesem Dialog wählen Sie zuerst den passenden Projekttyp aus. Für Webanwendungen eignen sich die Einträge VISUAL BASIC-PROJEKTE und VISUAL C#-PROJEKTE, abhängig davon, welche Sprache Sie verwenden wollen. Dann wählen Sie den Eintrag ASP.NET-Webanwendung und geben den Namen des Projekts ein. Der Name wird automatisch auch als Name des Webordners verwendet. Für den Speicherort können Sie eine gültige URL zu einem Webserver eingeben, auf dem das .NET-Framework installiert ist. Normalerweise verwenden Sie wohl dazu den lokalen Webserver, da Sie Ihre Webanwendungen auf einem separaten Entwicklungsrechner entwickeln (sollten). Sie können das Projekt aber auch ohne Probleme auf einem anderen Server erzeugen. Voraussetzung dafür ist, dass die Version des .NET-Framework auf dem anderen Rechner mit der lokalen Version identisch ist.

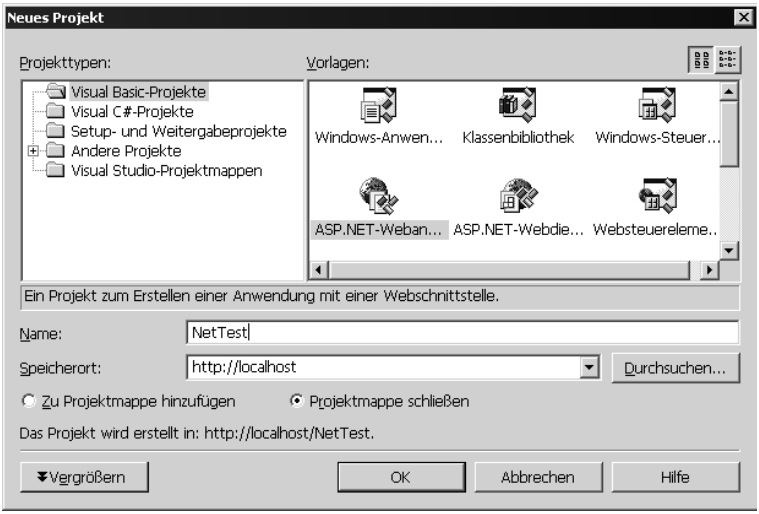


Bild 12.2: Der Dialog zum Erzeugen eines neuen Projekts

Wenn Sie als Speicherort nur den Namen des Webservers angeben, wird der Webordner im Stammordner des Servers erzeugt. Alternativ können Sie auch einen (noch nicht vorhandenen) Unterordner angeben (z. B. *http://localhost/Shop/Admin*, wenn der Ordner SHOP noch gar nicht existiert). Sie können aber auch einen vorhandenen (auch virtuellen) Webordner als Speicherort verwenden. Visual Studio sorgt automatisch dafür, dass der Zielordner erzeugt und als Webanwendung konfiguriert wird, falls dies noch nicht der Fall sein sollte.

Wenn Sie den Dialog dann bestätigen, legt Visual Studio den Webordner automatisch an, konfiguriert ihn und erzeugt gleich noch einige Dateien in diesem Ordner.

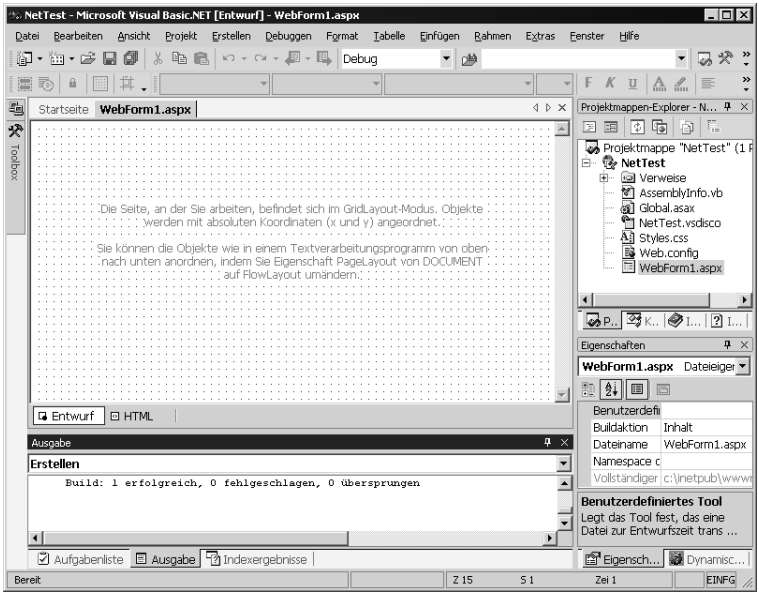


Bild 12.3: Visual Studio mit einem neuen Webprojekt

Der Projektmappen-Explorer und die Dateien des Projekts

Nachdem Sie ein neues Projekt erzeugt haben, sehen Sie die Dateien des Projekts im Projektmappen-Explorer. Dieser verwaltet die Verweise Ihres Projekts (zu externen Assemblierungen) und bildet die Dateien ab, die zum Webprojekt gehören. Hier erhalten Sie per Doppelklick schnellen Zugriff auf die Dateien, können diese entfernen und umbenennen. Die erzeugten Dateien erläutert die folgende Auflistung:

- ASSEMBLYINFO.VB: In dieser Datei können Sie allgemeine Informationen zur später erzeugten Assemblierung für Ihr Projekt unterbringen. Dazu gehören z. B. der Titel der Assemblierung, eine Beschreibung, das Copyright und die Versionsnummer. Die hier eingegebenen Daten werden beim Kompilieren in die Metadaten der Assemblierung übernommen.
- GLOBAL.ASAX: Die globale Datei für die Deklaration von Session- und Application-Ereignisprozeduren und für statische, globale Objektdeklarationen (vgl. Kapitel 8)

- *Projektname*.VSDISCO: In dieser Datei verwaltet die Entwicklungsumgebung die Ordner, die zwar im Webordner gespeichert sind, aber nicht zum eigentlichen Projekt gehören. Dazu gehören per Voreinstellung die Ordner, die die FrontPage-Servererweiterungen verwenden.
- STYLES.CSS: Eine normale CSS-Datei, die Sie verwenden können, um die Stile Ihrer Webanwendung zu verwalten und mit `<link rel="stylesheet" type="text/css" href="Styles.css">` in einzelne aspx-Seiten zu verknüpfen
- WEB.CONFIG: Die Stammkonfigurationsdatei der Webanwendung (vgl. Kapitel 11)
- WEBFORM1.ASPX: Eine leere ASPX-Datei

Eine Datei ist allerdings zunächst nicht sichtbar: Visual Studio-ASPX-Seiten verwenden Code-Behind-Klassen. Die `aspx`-Datei ist deshalb automatisch mit einer Quellcode-Datei verknüpft, die die Code-Behind-Klasse enthält. Diese Datei besitzt denselben Namen wie die ASPX-Datei mit der zusätzlichen Endung .VB (Visual Basic) oder .CS (C#). Sie öffnen diese Datei, indem Sie das Symbol CODE ANZEIGEN (ganz links) in der Symbolleiste des Projektmappen-Explorers anklicken. Alternativ können Sie auch das Symbol ALLE DATEIEN ANZEIGEN anklicken (zweites Symbol von rechts), um die Code-Behind-Datei als Untereintrag der ASPX-Datei zu sehen (Abbildung 12.4).

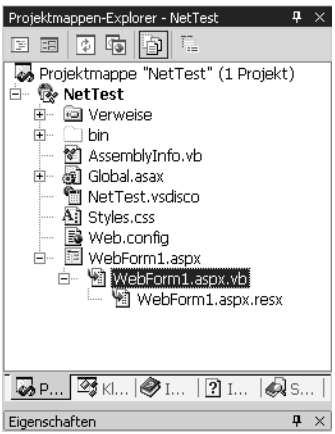


Bild 12.4: Der Projektmappen-Explorer mit der Anzeige aller Dateien des Projekts

Umbenennen des Webformulars

Vor dem Programmieren sollten Sie als Erstes das Webformular umbenennen. Dummerweise müssen Sie den Dateinamen der ASPX-Datei *und* den Klassennamen der Code-Behind-Klasse separat ändern (in der CLASS-Anweisung der Code-Behind-Datei und im `inherits`-Attribut der `@Page`-Anweisung der ASPX-Datei), wenn alle Namen konsistent sein sollen. Ich gehe deshalb lieber einen anderen Weg: Entfernen Sie die ASPX-Datei aus dem Projekt (Rechtsklick auf der Datei | LÖSCHEN). Erstellen Sie eine neue Datei (Menü PROJEKT | WEB FORM HINZUFÜGEN). Geben Sie dieser Datei beim Erstellen einen aussagekräftigen Namen (z. B. DEFAULT.ASPX, weil es ja wahrscheinlich auch bei Ihnen die Startseite ist). Stellen Sie diese Datei dann noch als Startseite für das Projekt ein (Rechtsklick auf der Datei | ALS STARTSEITE FESTLEGEN). Visual Studio startet die Anwendung immer mit der so eingestellten Startseite, wenn Sie F5 oder den entsprechenden Menübefehl zum Starten (Debuggen) betätigen.

12.2 Die Fenster der Entwicklungsumgebung

Die Entwicklungsumgebung besitzt eine große Anzahl Fenster für die verschiedensten Aufgaben. Ich beschreibe hier die wichtigsten. Den Projektmappen-Explorer kennen Sie ja bereits.

Falls ein Fenster einmal nicht angezeigt wird, rufen Sie einfach den passenden Befehl im ANSICHT*-Menü auf.*

Der Code-/Designer-Bereich

In der Mitte der Entwicklungsumgebung finden Sie den Code-/Designer-Bereich. Hier öffnet Visual Studio Fenster, in denen Sie den Programmcode oder die Oberfläche einer Datei bearbeiten können. Eine ASPX-Seite kann z. B. in der HTML-Code-Ansicht oder in der Entwurfs-Ansicht geöffnet werden. Sie können die Seite in beiden Ansichten bearbeiten. Beachten Sie, dass Visual Studio zwei Design-Modi unterscheidet: Im FlowLayout-Modus legen Sie die Steuerelemente fließend hintereinander an wie in einem normalen HTML-Dokument, wenn Sie keine CSS-Positionierung verwenden. Im GridLayout-Modus verwendet der Designer allerdings absolute CSS-Positionierun-

gen, sodass Sie die Steuerelemente frei auf der Seite platzieren können. Sie stellen dieses Verhalten über die Eigenschaft `pageLayout` der Seite.

Die Toolbox

Ganz links finden Sie die Toolbox. Wenn Sie die Maus auf diese Box bewegen, klappt das Toolbox-Fenster auf.

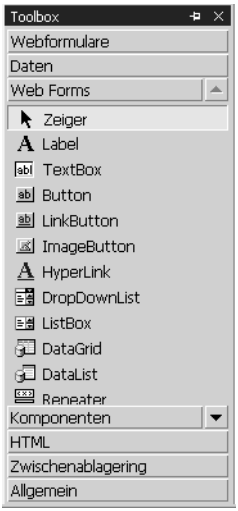


Bild 12.5: Die Visual Studio-Toolbox

Hier finden Sie eine große Anzahl Steuerelemente und andere Komponenten. Nicht alle können Sie in Web-Projekten einsetzen. Das Register KOMPONENTEN beinhaltet Komponenten, deren Einsatz nur in Windowsanwendungen Sinn macht. Für Webanwendungen sind die Register WEB FORMS und HTML am wichtigsten. Auf diesen Registern finden Sie die verfügbaren Web- und HTML-Steuerelemente. Wenn Sie eines dieser Steuerelemente verwenden wollen, ziehen Sie dieses einfach auf ein Webformular. Dort können Sie das Steuerelement dann mit der Maus positionieren und in der Größe verändern und dessen Eigenschaften einstellen. Das Register DATEN enthält Komponenten, die zum Datenzugriff verwendet werden. Alternativ zur Erzeugung von Datenzugriffsobjekten im Quellcode können Sie auch diese Komponenten verwenden, die Sie ähnlich einem Steuerele-

ment auf eine Seite ziehen und dort einstellen können. Diese Komponenten sind aber bei der Ausführung der Seite nicht sichtbar.

Das Eigenschaftenfenster

Im Eigenschaftenfenster (unter dem Projektmappen-Explorer) stellen Sie die Eigenschaften des gerade aktiven Objekts ein. Besonders für Webanwendungen ist dieses Fenster interessant. Aktivieren Sie z. B. ein Webformular, können Sie alle Eigenschaften der Seite recht konsistent im Eigenschaftenfenster einstellen, unabhängig davon, ob es sich um Attribute der @Page-Anweisung oder um Werte der Standard-Tags der Seite handelt. Der Titel der Seite (der im `title`-Tag gespeichert wird) wird z. B. in der `title`-Eigenschaft verwaltet. Verwenden Sie wenn möglich immer das Eigenschaftenfenster zur Einstellung der Eigenschaften, dann müssen Sie nicht umständlich im Quelltext der Seite suchen. Änderungen im Quelltext werden aber auch sofort im Eigenschaftenfenster dargestellt. Aktivieren Sie immer zuerst das Objekt, dessen Eigenschaften Sie einstellen wollen. Das Eigenschaftenfenster funktioniert natürlich auch für HTML- und Web-Steuerelemente.

Die dynamische Hilfe

Das Fenster für die dynamische Hilfe befindet sich ebenfalls unter dem Projektmappen-Explorer. Sie müssen normalerweise erst das Register Dynamische Hilfe anklicken, um dieses Fenster anzuzeigen. Die dynamische Hilfe ist sehr hilfreich. Setzen Sie den Eingabecursor im Quellcode in ein Schlüsselwort und schon zeigt die dynamische Hilfe (meist) Links zu passenden Hilfethemen an. Probieren Sie es einfach aus.

12.3 Die wichtigen Eigenschaften der Entwicklungsumgebung und des Projekts

Bevor Sie beginnen zu programmieren, sollten Sie die Eigenschaften der Entwicklungsumgebung und des Projekts überprüfen.

Eigenschaften der Entwicklungsumgebung

Die Eigenschaften der Entwicklungsumgebung erreichen Sie über das Menü Extras | Optionen. Wichtige Eigenschaften finden Sie hier im Eintrag Text-Editor. Hier können Sie u. a. die Tabulatorgröße für einzelne oder auch für alle Sprachen definieren.

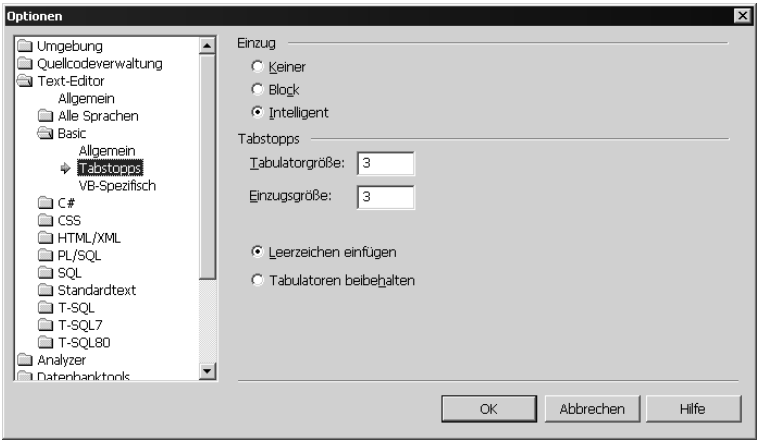


Bild 12.6: Die Optionen von Visual Studio

Projekteigenschaften

Wichtiger als die Eigenschaften der Entwicklungsumgebung sind die Projekteigenschaften. Diese erreichen Sie, indem Sie den Befehl Eigenschaften im Projekt-Menü betätigen. Sehr wichtige Einstellungen hier sind die, die Sie unter dem Eintrag Allgemeine Eigenschaften | Erstellen finden.

Für Visual Basic-Projekte sollten Sie diese Eigenschaften so einstellen, wie in Abbildung 12.7 gezeigt (in C# gibt es diese Eigenschaften nicht).

Option Explicit bewirkt, dass Sie Variablen deklarieren müssen. Ist diese Eigenschaft ausgeschaltet, müssen Variablen nicht deklariert werden, sondern werden vom Compiler einfach bei der ersten Benutzung implizit deklariert. Wenn Sie in Ihrem Projekt keine sehr schwer lokalisierbaren logischen Fehler – aufgrund der falschen Schreibweise eines Variablennamens – erzeugen wollen, schalten Sie diese Eigenschaft ein.

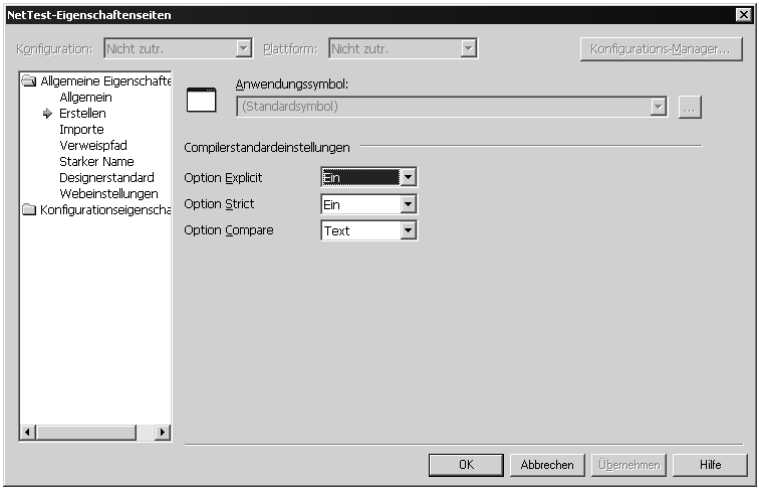


Bild 12.7: Die Projektoptionen eines Visual Basic-Webanwendungs-Projekts

Option Strict zwingt Sie dazu, Datentypen immer passend zu verwenden. Schalten Sie diese Eigenschaft aus, können Sie z. B. einer Integer-Variable auch einen String zuweisen. Der Compiler konvertiert den String automatisch in einen Integer-Wert, falls dies möglich ist. Falls nicht, erzeugt Ihr Programm dann eine Ausnahme (einen Laufzeitfehler). Leider bewirkt dieses Relikt aus alten Visual Basic-Tagen auch, dass Sie versehentlich einen falschen Datentyp einsetzen können. Typsicherheit ist einer der großen Vorteile von guten Programmiersprachen (wie C#, C++, Delphi). Schalten Sie diese Eigenschaft also besser ein.

Option Compare bewirkt mit der Einstellung Binär, dass der Compiler bei Textvergleichen Groß- und Kleinschreibung unterscheidet (»a« ist ungleich »A«). Da dies in normalen Anwendungen eher hinderlich ist, setzen Sie diese Eigenschaft besser auf den Wert Text.

12.4 ASP.NET-Anwendungen mit Visual Studio.NET entwickeln

Die Entwicklung von Webanwendungen mit Visual Studio.NET ist recht einfach. Ein großer Vorteil gegenüber einem normalen Editor ist z. B., dass Sie die Oberfläche der Anwendung mit der Maus gestalten können.

Gestaltung der Oberfläche

Die Oberfläche einer Anwendung können Sie entweder im HTML-Quelltext oder in der Entwurfsansicht der ASPX-Seite gestalten. Im HTML-Quelltext müssen Sie die entsprechenden Tags aber selbst eingeben. Deswegen empfiehlt sich immer, den ersten Entwurf in der Entwurfsansicht der Seite zu gestalten. Ziehen Sie dazu einfach die benötigten Steuerelemente auf die Seite. Sie können die Seite in zwei Layout-Varianten gestalten: Im GridLayout-Modus werden alle Steuerelemente über absolute Positionsangaben (x, y) positioniert, im FlowLayout-Modus erfolgt die Positionierung ähnlich wie bei einer normalen HTML-Seite, Element hinter Element. Sie stellen diesen Modus über die Eigenschaft `pageLayout` der Seite um.

Wenn Sie die Eigenschaften der Seite ändern wollen, reicht normalerweise ein Klick auf einen freien Bereich der Seite, um das Eigenschaftenfenster auf die Seite einzustellen. Manchmal funktioniert das aber nicht, z. B. wenn ein `p`*-Tag in der* ASPX*-Seite verwendet wird und Sie auf diesen Absatz klicken. Wählen Sie dann den Eintrag* DOCUMENT *aus der Liste des Eigenschaftenfensters aus, um die Eigenschaften der Seite anzuzeigen.*

Ziehen Sie anschließend die benötigen Steuerelemente auf die Seite und passen diese mit Hilfe der Maus und über das Eigenschaftenfenster an. Abbildung 12.8 zeigt einen Ausschnitt einer Seite, auf der ein Label und ein Schalter platziert wurden.

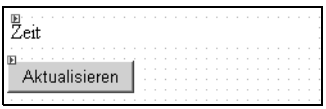


Bild 12.8: Ein Label und ein Schalter auf einer aspx-Seite im Entwurf

Die einzelnen Steuerelemente erläutere ich kurz im nächsten Kapitel.

Die über das Entwurfsfenster (mit der Maus) oder im Eigenschaftenfenster eingestellten Eigenschaften (im Beispiel sind das die Position, die Größe, der Name und der Text der Steuerelemente) werden von Visual Studio als Attribut der HTML-Tags eingetragen:

```
<form id="Form1" method="post" runat="server">
   <asp:button id="btnAktualisieren" style="Z-INDEX: 101;
    LEFT: 21px; POSITION: absolute; TOP: 58px"
    runat="server" Text="Aktualisieren" Width="99px"
    Height="25px"></asp:button>
   <asp:label id="lblZeit" style="Z-INDEX: 102; LEFT:
    24px; POSITION: absolute; TOP: 26px" runat="server"
    Width="239px" Height="19px">Zeit</asp:label>
</form>
```

Änderungen dieser Attribute können Sie auch im HTML-Quelltext vornehmen.

Mit Strg + F5 können Sie das Projekt (ohne Debuggen) starten, um die Oberfläche im Browser zu begutachten.

Vergeben Sie den Steuerelementen vor der Programmierung in deren `Name`*-Eigenschaft einen eindeutigen Namen. Visual Studio erzeugt Ereignisprozeduren immer per Voreinstellung unter Verwendung des Namens des Steuerelements, wenn Sie doppelt auf ein Steuerelement klicken. Wenn Sie den Namen dann später ändern, behält die Ereignisprozedur ihren alten Namen und die Benennung wird inkonsistent.*

Bearbeitung des Quellcodes

Die Bearbeitung des Quellcodes ist in Visual Studio sehr einfach. Wenn Sie ein Ereignis der Seite oder eines Steuerelements auswerten wollen, klicken Sie einfach doppelt auf das Objekt. Visual Studio erzeugt dann automatisch eine Ereignisprozedur für das Standard-Ereignis (das Ereignis, welches am häufigsten verwendet wird). Die Ereignisprozedur wird in der Code-Behind-Klasse der Seite erzeugt.

Der Quellcode der Seite sieht dann prinzipiell so aus, wie ich es bereits für Code-Behind-Klassen in Kapitel 11 beschrieben habe.

```
Public Class CDefault
    Inherits System.Web.UI.Page
    Protected WithEvents btnAktualisieren As _
       System.Web.UI.WebControls.Button
    Protected WithEvents lblZeit As _
       System.Web.UI.WebControls.Label

 Web Form Designer Generated Code 

    Private Sub Page_Load(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles MyBase.Load
        ' Hier Benutzercode zur Seiteninitialisierung einfügen
        lblZeit.Text = DateTime.Now.ToLongTimeString
    End Sub

    Private Sub btnAktualisieren_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnAktualisieren.Click
        lblZeit.Text = DateTime.Now.ToLongTimeString
    End Sub
End Class
```

Bild 12.9: Quellcode einer Code-Behind-Klassendatei in Visual Studio

Neu ist hier die *Region* WEB FORM DESIGNER GENERATED CODE. Regionen können Sie selbst auch verwenden, indem Sie diese mit `#Region` *Name* und `#End Region` definieren. Regionen sollen einfach den Quelltext übersichtlicher machen, weil Sie diese über das Plus-Symbol aufklappen und über das Minus-Symbol (wieder) zuklappen können. Aufgeklappt sieht diese Region dann so aus wie in Abbildung 12.10.

```
#Region " Web Form Designer Generated Code "

    'Dieser Aufruf ist für den Webformular-Designer erforderlich.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles MyBase.Init
        'CODEGEN: Diese Methode ist für den Webformular-Designer erforderlich
        'Verwenden Sie nicht den Code-Editor zur Bearbeitung.
        InitializeComponent()
    End Sub

#End Region
```

Bild 12.10: Die aufgeklappte Region für den internen Code des Webform-Designers

Diese spezielle Region enthält Methoden, die vom Webform-Designer (das ist ein anderer Begriff für das Entwurfsfenster) verwendet werden. Wenn Sie Komponenten (z. B. zum Datenbankzugriff) auf der Seite verwenden, erzeugt und initialisiert der Designer die dazugehörigen Objekte in der Methode `InitializeComponent`, die in der Ereignisprozedur des `Init`-Ereignisses der Seite aufgerufen wird. Sie sollten den Quellcode dieser Region eigentlich nicht verändern. Wenn Sie genau wissen, was Sie machen, können Sie dort aber schnell die Eigenschaftswerte einer Komponente ändern.

Neu ist auch, dass Visual Studio die Ereignisprozeduren mit dem Zusatz `Handles` *Objektname.Ereignisname* kennzeichnet. Dieser Zusatz teilt dem Compiler mit, dass diese Prozedur das angegebene Ereignis für das angegebene Objekt behandelt. Wenn Sie ASP.NET-Seiten ohne Visual Studio erzeugen, können Sie auf diesen Zusatz verzichten, wenn das Attribut `AutoEventWireup` der `@Page`-Anweisung auf "true" steht (was per Default der Fall ist). Der Compiler sorgt dann dafür, dass die Ereignisprozeduren an deren Namen erkannt werden (`Public Sub` *Objektname_Ereignisname*). Visual Studio stellt `AutoEventWireup` immer auf "false", womit diese automatische Erkennung (die Performance kostet) nicht verwendet wird.

Der Rest in einfach: Innerhalb der Klasse können Sie programmieren und programmieren und ...

Das soll hier kein Thema sein. Lesen Sie u. U. das Buch zu C# oder Visual Basic.NET aus der Nitty-Gritty-Reihe. Dort ist alles weitere beschrieben.

13 Serverseitige ASP.NET-Steuerelemente

Das .NET-Framework definiert im `System.Web`-Namensraum eine große Anzahl serverseitig ausgeführter Steuerelemente, die in diesem Kapitel behandelt werden. Bevor ich aber auf die einzelnen Steuerelemente eingehe, beschreibe ich zunächst einige wichtige Grundlagen.

13.1 Grundlagen

Serverseitige versus HTML-Steuerelemente

Wie Sie in Kapitel 11 ja bereits erfahren haben, können Sie auf einer ASPX-Seite sowohl einfache HTML-Steuerelemente als auch serverseitige Steuerelemente einsetzen. Serverseitige Steuerelemente gibt es meist noch in zwei Varianten: in der HTML- und der speziellen ASP-Version. Eine Textbox können Sie z. B. einfach nur mit HTML erzeugen:

```
<input type="text" id="txtDemo1">
```

oder als serverseitiges HTML-Steuerelement, indem Sie das Attribut `runat` mit dem Wert "server" hinzufügen:

```
<input type="text" id="txtDemo2" runat="server">
```

Die dritte Alternative ist die Erstellung als serverseitiges ASP-Steuerelement:

```
<asp:TextBox id="txtDemo3" runat="server"></asp:TextBox>
```

Serverseitige Steuerelemente sind mit dem Attribut `runat="server"` gekennzeichnet. Diese Steuerelemente sind Objekte innerhalb der Seite. Das `Page`-Objekt fordert diese Objekte bei seiner Ausführung (nach der Anforderung der ASPX-Seite durch einen Browser) auf, passenden HTML-Code zu erzeugen. Das allein macht aber noch nicht den Sinn serverseitig ausgeführter Steuerelemente aus.

Weil diese Steuerelemente Objekte innerhalb der Seite sind, kann das `Page`-Objekt Ereignisse auswerten, die für die Steuerelemente an-

liegen. Wie das prinzipiell funktioniert, habe ich ja bereits in Kapitel 11 behandelt.

Daneben können Sie bei serverseitigen Steuerelementen im Programmcode der Seite auf deren Eigenschaften und Methoden zugreifen. Für normale HTML-Steuerelemente ist das alles nicht möglich.

Normale HTML-Steuerelemente besitzen aber auch eine Bedeutung. Sie werden verwendet:

- wenn ein Element nur zur Anzeige statischer Daten oder als einfacher Hyperlink eingesetzt wird,
- wenn ein Steuerelement nur dazu verwendet wird, ein clientseitiges Script auszuführen,
- wenn das Element ein Submit-Button zum klassischen Absenden eines Formulars ist (das dann im `Load`-Ereignis der Zielseite über `Request.Form` ausgewertet wird).

Einfache HTML-Steuerelemente benötigen wesentlich weniger Ressourcen auf dem Server und werden schneller ausgeführt als serverseitige Steuerelemente. Um Ihre Seiten möglichst performant zu gestalten, sollten Sie serverseitige Steuerelemente nur dann einsetzen, wenn Sie diese auch benötigen (d. h. wenn Sie deren Ereignisse auswerten oder im Programm auf deren Eigenschaften zugreifen wollen).

Wo liegt nun aber der Unterschied zwischen den serverseitigen HTML- und ASP-Steuerelementen? Die Antwort ist einfach: Die HTML-Steuerelemente sind inkonsistent. Einige der Eigenschaften dieser Steuerelemente werden in Attributen angegeben, andere wieder in Style-Angaben. Einige Elemente, wie das `textarea`- und das `div`-Element, speichern Eigenschaften (ihren Inhalt) zwischen dem Start- und dem Ende-Tag. Für prinzipiell gleiche Aufgaben werden oft ganz unterschiedliche Steuerelemente eingesetzt. Eine einfache Textbox wird zum Beispiel über den `input`-Tag eingerichtet, eine mehrzeilige Textbox dagegen über den `textarea`-Tag. Der `input`-Tag ist gleich mit mehreren Funktionen belegt, u. a. wird damit ja auch ein Button eingerichtet. Der Umgang mit diesen Elementen ist für den Entwickler deshalb inkonsistent und kompliziert.

Die ASP-Steuerelemente lösen dieses Problem, indem sie einen konsistenten Satz an Steuerelementen mit immer wieder gleichen Eigenschaften anbieten. Das `Textbox`-Steuerelement und das `Label` besitzen zum Beispiel beide die Eigenschaft `Text`, die den Inhalt des Elements speichert. Das `Textbox`-Steuerelement erzeugt bei seiner Ausführung je nach Einstellung der Eigenschaft `TextMode` ein `input`-Element mit `type="text"` oder `type="password"` oder ein `textarea`-Element. Im Programm greifen Sie aber immer auf die Eigenschaft `Text` zu, wenn Sie den Inhalt des Elements bearbeiten wollen. Andere Eigenschaften sind genauso konsistent. Sie müssen sich nicht mehr mit den unterschiedlichen Tags und deren Attributen herumschlagen, sondern verwenden einfach das passende Steuerelement.

Die ASP-Steuerelemente besitzen aber noch einen weiteren Vorteil: Sie verwenden die im `Page`-Objekt verfügbaren Informationen über den verwendeten Browser, um browserspezifischen Quellcode zu erzeugen.

Verwenden Sie für Steuerelemente, die Sie im Quellcode bearbeiten wollen, möglichst ASP-Steuerelemente. Ihr Quellcode wird damit wesentlich konsistenter. Achten Sie aber auf die Performance. HTML-Elemente mit statischem Inhalt, die nicht im Programmcode bearbeitet werden sollen, sollten Sie als HTML-Steuerelement anlegen. Müssen Sie später doch im Quellcode auf diese Elemente zugreifen, können Sie diese recht einfach über die Eigenschaft `runat` *in ein serverseitig ausgeführtes HTML-Steuerelement umwandeln. Mit ein wenig Arbeit im Quellcode machen Sie daraus aber auch ein ASP-Steuerelement.*

Einstellung der Eigenschaften der Steuerelemente

Innerhalb von Visual Studio können Sie die Eigenschaften der Steuerelemente sehr einfach über das Eigenschaftenfenster einstellen. Wenn Sie aber einen einfachen Editor für die Erstellung von `aspx`-Seiten verwenden, müssen Sie die Werte der Eigenschaften »von Hand« zuweisen. Bei den HTML-Steuerelementen geht das ganz einfach wie bei HTML selbst (Kapitel 4). Beachten Sie, dass einige Eigenschaften über Stile definiert werden (Kapitel 5). Eine Textbox mit einem blauen

Hintergrund, einem Rahmen und einer definierten Größe wird so deklariert:

```
<input type="text" id="txtDemo1"
 style="border: black thin solid; width: 100px;
 height: 22px; background-color: #3300FF">
```

Die Einstellung der ASP-Steuerelemente ist wesentlich einfacher, weil die meisten Eigenschaften konsistent über Attribute eingestellt werden. Eine ASP-Textbox wie im vorherigen Beispiel wird folgendermaßen definiert:

```
<asp:TextBox Id="txtDemo2" Runat="server"
 Height="22px" Width="100px" BackColor="#3300FF"
 BorderColor="#000000" BorderStyle="Double"></asp:TextBox>
```

Leider sind die meisten Stile davon ausgenommen. Diese werden auch bei ASP-Steuerelementen über den Style-Tag gesetzt.

Die Standardeigenschaft eines ASP-Steuerelements können Sie auch als Wert des ASP-Elements angeben. Bei einer Textbox können Sie z. B. statt:

```
<asp:Textbox Id="txtName" Text="Zaphod" Runat="server"/>
```

auch:

```
<asp:Textbox Id="txtName"
 Runat="server"/>Zaphod</asp:Textbox>
```

schreiben.

Wenn Sie im Programm auf die Eigenschaften serverseitiger Steuerelemente zugreifen wollen, müssen Sie bei HTML-Steuerelementen beachten, dass viele der Eigenschaften über Stile verwaltet werden. Normale Eigenschaften sind benannt wie die entsprechenden HTML-Attribute. Diese Eigenschaften können Sie direkt bearbeiten. Das folgende Beispiel schreibt einen neuen Text in die Textbox:

```
txtDemo1.Value = "Das ist ein Test"
```

Stile müssen Sie über die `Style`-Eigenschaft definieren, die eine Auflistung der Stile des Elements ist. Als Schlüssel geben Sie den Namen des Stils an. Das folgende Listing definiert die Vordergrund- und die Hintergrundfarbe einer Textbox neu:

```
txtDemo1.Style("color") = "white"
txtDemo1.Style("background-color") = "red"
```

Zusätzlich müssen Sie beachten, dass Sie die Werte in die Attribute eintragen müssen, die in HTML verwendet werden. Farben werden in HTML über Zeichenketten definiert, die festgelegte Farbnamen oder Farbwerte in hexadezimaler Form enthalten.

Bei ASP-Steuerelementen ist der Zugriff im Programm einfacher, weil Sie meist auf Eigenschaften zugreifen:

```
txtDemo2.Text = "Das ist ein Text"
txtDemo2.ForeColor = System.Drawing.Color.White
txtDemo2.BackColor = System.Drawing.Color.Red
```

Die Werte, die Sie in die Eigenschaften schreiben, sind meist über Konstanten in einer Klasse des .NET-Framework definiert. Das Beispiel verwendet z. B. die Konstanten der Klasse `System.Drawing.Color` für die Angabe der Farben.

Cross-Browser-Seiten und das Ziel-Schema

Wenn Sie mit Visual Studio arbeiten, können Sie in der Eigenschaft `targetSchema` einer Seite festlegen, wie Visual Studio den Quelltext einer Seite erzeugt. Hier können Sie z. B. bestimmen, dass HTML 3.2-kompatibler Quellcode erzeugt werden soll. Das Ziel-Schema geben Sie in der Eigenschaft `targetSchema` der Seite an. Möglich sind zurzeit die folgenden Einstellungen:

- Internet Explorer 3.02 / Navigator 3.0: erzeugt reinen HTML 3.2-Code ohne CSS. Positionierungen werden von Visual Studio über Tabellen realisiert,
- Navigator 4.0: erzeugt HTML 4-Code mit CSS für den Netscape Navigator ab Version 4.0,
- Internet Explorer 5.0: erzeugt HTML 4-Code mit CSS für den Internet Explorer Version 5.0.

Je nach Einstellung erzeugt der Entwurfs-Designer von Visual Studio für absolute Positionierungen Style-Angaben (Netscape 4 und Internet Explorer 5) oder positioniert über geschachtelte Tabellen (Internet Explorer 3.02 / Navigator 3.0). Außerdem werden automatisch für die aktuelle Einstellung nicht verfügbare Features in der Entwick-

lungsumgebung ausgeblendet. Visual Studio stellt mit diesem Feature eine Hilfestellung zum Erzeugen kompatiblen Quellcodes zur Verfügung.

Beachten Sie, dass die Einstellung des Ziel-Schemas nicht die Erzeugung des browserspezifischen Quellcodes durch die Steuerelemente betrifft. Diese überprüfen per Voreinstellung immer, welche Möglichkeiten der Browser besitzt, und erzeugen automatisch passenden HTML-Code. Dieses Verhalten können Sie wohl in der Eigenschaft `clientSchema` *beeinflussen.*

Eigenartig ist, dass Sie das Ziel-Schema zurzeit (noch) nicht für den Internet Explorer 4, den Internet Explorer 6 und den Netscape 6 einstellen können und dass die Einstellung scheinbar für jede Seite separat vorgenommen werden muss.

Später werden wohl noch Ziel-Schemata für Endgeräte wie WAP-Handys oder UMTS-Geräte hinzukommen.

13.2 Die HTML-Steuerelemente

Die HTML-Steuerelemente repräsentieren HTML-Elemente auf der ASPX-Seite. Mit diesen Steuerelementen können Sie recht flexibel programmieren. So können Sie die Eigenschaften der Elemente einer Seite im Programm auswerten oder verändern und neue Elemente dynamisch erzeugen. Die HTML-Steuerelemente besitzen außerdem einige Ereignisse, die Sie im Quellcode abfangen können. Voraussetzung für die Auswertung der Eigenschaften und der Ereignisse ist, dass die Steuerelemente serverseitig ausgeführt werden.

Da Sie zum programmgesteuerten Zugriff besser die ASP-Steuerelemente verwenden sollten (siehe ab Seite 368), beschreibe ich die HTML-Steuerelemente hier nur grundlegend.

13.2.1 Die Basisklassen HtmlControl und HtmlContainerControl

Die Klasse HtmlControl

Die HTML-Steuerelemente, die im Namensraum `System.Web.UI.HtmlControls` definiert sind, sind alle von der Basisklasse `HtmlControl` ab-

geleitet, besitzen also immer die Eigenschaften, Methoden und Ereignisse dieser Klasse. Die wichtigsten Eigenschaften von `HtmlControl` finden Sie in Tabelle 13.1, die Methoden in Tabelle 13.2.

Eigenschaft	Bedeutung
`Controls`	gibt eine `ControlCollection`-Auflistung zurück mit Verweisen auf alle Steuerelemente, die auf dem Steuerelement angelegt sind
`Disabled`	bestimmt, ob das Steuerelement aktiviert oder deaktiviert ist
`EnableViewState`	Über diese Eigenschaft können Sie bestimmen, ob das Steuerelement seinen Status auf der Seite im Viewstate speichert
`ID`	die ID des Steuerelements
`Page`	eine Referenz zu der Seite, die das Steuerelement enthält
`Parent`	eine Referenz auf das Steuerelement, in dem das Steuerelement eingebettet ist (sofern dies der Fall ist)
`Style`	eine Auflistung aller Stile des Steuerelements
`TagName`	der Name des HTML-Tags (z. B. "a" für einen Anker oder "div" für ein `div`-Element)
`Visible`	definiert, ob das Steuerelement sichtbar ist

Tabelle 13.1: Die wichtigsten Eigenschaften der Basisklasse HtmlControl

Methode	Beschreibung
`DataBind`	Diese Methode wird verwendet, wenn ein Steuerelement an eine Datenquelle angebunden wurde, um die Daten anzuzeigen. Datenbindung wird in Kapitel 14 behandelt.
`FindControl`	Über diese Methode können Sie den Behälter des Steuerelements nach einem anderen Steuerelement durchsuchen.
`HasControls`	gibt `True` zurück, wenn das Steuerelement selbst ein Behälter für andere Steuerelemente ist

Tabelle 13.2: Die Methoden der Basisklasse HtmlControl

Die Klasse HtmlContainerControl

Die Klasse `HtmlContainerControl` ist von `HtmlControl` abgeleitet und dient als Basisklasse für alle HTML-Elemente, die einen schließenden Tag besitzen (also Behälter für andere Elemente sein können). Diese Klasse besitzt zusätzlich die in Tabelle 13.3 aufgelisteten Eigenschaften.

Eigenschaft	Bedeutung
`InnerHtml`	der HTML-Quellcode des Inhalts des Elements
`InnerText`	der reine Text des Inhalts des Elements

Tabelle 13.3: Die Eigenschaften der Basisklasse HtmlContainerControl

13.2.2 Übersicht über die Steuerelemente

Die HTML-Steuerelemente werden in ASP.NET über Klassen repräsentiert, die Sie im Namensraum `System.Web.UI.HtmlControls` finden. Ich stelle diese Steuerelemente hier nur kurz vor, weil Kapitel 4 die HTML-Elemente ja bereits behandelt hat. Die linke Spalte der Tabelle zeigt den Namen des Steuerelements, wie er in der Toolbox von Visual Studio auftaucht. Die angegebenen Klassen können Sie verwenden, wenn Sie diese Steuerelemente dynamisch erzeugen wollen (siehe Seite 393).

Steuerelement	Klasse	Beschreibung / HTML-Darstellung
Generic (nicht in der Toolbox von Visual Studio)	HtmlGenericControl	Ein generisches Steuerelement kann jeden beliebigen Tag annehmen. Solch ein Steuerelement können Sie verwenden, um beliebige HTML-Elemente dynamisch zu erzeugen (siehe Seite 393). Dieses Steuerelement erlaubt als einziges das Beschreiben der Eigenschaft `TagName`.

Steuer-element	Klasse	Beschreibung / HTML-Darstellung
Anchor (nicht in der Toolbox von Visual Studio)	`HtmlAnchor`	ein Anker: `<a></a>`
Form	`HtmlForm`	ein Formular: `<form></form>`
Label	`HtmlGenericControl` mit `TagName` = "div"	ein `div`-Element: `<div></div>`
Button	`HtmlInputButton` mit `TagName` = "button" (im Konstruktor übergeben)	ein normaler Button: `<input type="button"/>`
Reset Button	`HtmlInputButton` mit `TagName` = "reset" (im Konstruktor übergeben)	ein Reset-Button: `<input type="reset"/>`
Submit Button	`HtmlInputButton` mit `TagName` = "submit" (im Konstruktor übergeben)	ein Submit-Button: `<input type="submit"/>`
Text Field	`HtmlInputText`	ein Textfeld: `<input type="text"/>`
Text Area	`HtmlTextArea`	ein mehrzeiliges Textfeld: `<textarea> </textarea>`
File Field	`HtmlInputFile`	ein Dateiauswahlfeld: `<input type="file"/>`
Password Field	`HtmlInputText` mit `TagName` = "password" (im Konstruktor übergeben)	ein Kennwortfeld: `<input type="password"/>`
Checkbox	`HtmlInputCheckBox`	eine Checkbox: `<input type="checkbox" />`

Steuerelement	Klasse	Beschreibung / HTML-Darstellung
Radio Button	`HtmlInputRadioButton`	ein RadioButton: `<input type="radio" />`
Hidden	`HtmlInputHidden`	ein verstecktes Feld: `<input type="hidden"/>`
Table	`HtmlTable`	eine Tabelle: `<table></table>`
	`HtmlTableRow`	eine Tabellenzeile: `<tr></tr>`
	`HtmlTableCell` mit `TagName = "th"` oder `"td"`	eine Tabellenzelle: `<td></td>` oder `<th></th>`
Flow Layout Panel	`HtmlGenericControl` mit `TagName = "div"`	ein `div`-Element mit dem ASP.NET-spezifischen Attribut `ms_positioning = "FlowLayout"`. Dieses Element dient als Container, auf dem Sie andere Elemente im Flow-Layout anlegen können.
Grid Layout Panel	`HtmlGenericControl` mit `TagName = "div"`	ein `div`-Element mit dem ASP.NET-spezifischen Attribut `ms_positioning = "GridLayout"`. Dieses Element dient als Container, auf dem Sie andere Elemente im Grid-Layout anlegen können.
Image	`HtmlImage`	ein Bild-Element: `<img/>`
Listbox	`HtmlSelect` mit `Size > 1`	ein mehrzeiliges Listenfeld: `<select size="n"></select>`
Dropdown	`HtmlSelect` mit `Size = 1`	ein einzeiliges Listenfeld: `<select size="1"></select>`

Steuer-element	Klasse	Beschreibung / HTML-Darstellung
Horizontal Rule	`HtmlGenericControl` mit `TagName = "hr"`	eine Linie: `<hr/>`

Tabelle 13.4: Übersicht über die HTML-Steuerelemente

13.2.3 Tipps und Tricks

Für die ASP-Steuerelemente beschreibe ich noch genauer, wie Sie damit im Programm arbeiten. Die HTML-Steuerelemente werden meist ähnlich verwendet. Deshalb zeige ich hier nur ein paar »Tipps« und »Tricks« zur Arbeit mit diesen Steuerelementen. Denken Sie daran, dass Sie für serverseitig ausgeführte Steuerelemente sowieso besser die ASP-Varianten verwenden sollten. Wenn Sie mit HTML-Steuerelementen programmieren wollen, müssen Sie diese auch serverseitig ausführen, dann können Sie auch gleich die ASP-Variante verwenden.

Füllen von Listenfeldern im Quellcode

Listenfelder können Sie recht schnell im Quellcode, z. B. in `Load`-Ereignis der Seite, füllen. Das Steuerelement muss dazu allerdings serverseitig ausgeführt werden, ansonsten haben Sie keinen Zugriff darauf. Erstellen Sie eine leere Liste:

```
<select id="lstDemo" size="4" runat="server"/>
```

Über die `Items`-Auflistung des Steuerelements können Sie dann einzelne Einträge anhängen:

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
 With lstDemo.Items
    .Add("Zaphod")
    .Add("Arthur")
    .Add("Ford")
    .Add("Trillian")
 End With
End Sub
```

13.3 Die ASP-Steuerelemente

Die ASP-Steuerelemente sind wesentlich flexibler und einfacher anzuwenden als die HTML-Steuerelemente. Ich beschreibe diese Steuerelemente deshalb wesentlich intensiver als deren HTML-Brüder und -Schwestern.

Die Steuerelemente, deren Verwendung nur mit Datenquellen Sinn macht (`DataGrid`*,* `DataList` *und* `Repeater`*), werden nicht in diesem Kapitel, sondern erst im nächsten beschrieben, weil dort die Datenbindung behandelt wird.*

13.3.1 Allgemeines

Ereignisse über AutoPostBack sofort auswerten

Normalerweise werden die Ereignisse der ASP-Steuerelemente erst dann ausgeführt, wenn der Anwender das Formular (z. B. über einen Submit-Button) bestätigt hat. Die meisten ASP-Steuerelemente sind aber in der Lage, auf Ereignisse direkt zu reagieren. Dieses Feature schalten Sie über die Eigenschaft `AutoPostBack` der Steuerelemente ein:

```
<asp:TextBox id="txtDemo" runat="server"
 AutoPostBack="True"></asp:TextBox>
```

Diese Eigenschaft ist natürlich nur bei Steuerelementen verfügbar, bei denen sie Sinn macht. Buttons besitzen `AutoPostBack` z. B. nicht, da die Betätigung eines Buttons sowieso zur Bestätigung des Formulars führt.

Für Steuerelemente, bei denen `AutoPostBack` eingeschaltet ist, erzeugt die Seite eine JavaScript-Funktion und den Aufruf dieser Funktion im entsprechenden HTML-Ereignis:

```
<form name="frmInput" method="post" action="Default.aspx"
 id="frmInput">
 <input type="hidden" name="__VIEWSTATE"
  value="dDwtMTM1ODU0MTM1Mzs7Pg==" />

 <input name="txtDemo1" type="text" id="txtDemo1"
  onchange="__doPostBack('txtDemo1','')"
```

```
  language="javascript" /><br>

<input name="txtDemo2" type="text" id="txtDemo2"
  onchange="__doPostBack('txtDemo2','')"
  language="javascript" /><br>

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
    var theform = document.frmInput;
    theform.__EVENTTARGET.value = eventTarget;
    theform.__EVENTARGUMENT.value = eventArgument;
    theform.submit();
 }
// -->
</script>
</form>
```

Wenn das HTML-Ereignis (im Browser) aufgerufen wird, führt das nun automatisch dazu, dass das Formular abgesendet wird. Und das bewirkt, dass das entsprechende ASP-Ereignis serverseitig ebenfalls ausgeführt wird.

Farbangaben

Farben für die ASP-Steuerelemente (und für andere .NET-Objekte) sind eigentlich immer ein Integerwert. Der Datentyp der Eigenschaften, die Farbwerte speichern, ist aber die Aufzählung `System.Drawing.Color`. Diese Aufzählung enthält Konstanten für die gängigen Farben, wie `Red`, `Green`, `Blue`, `Aqua` und `Tomato`. Wenn Sie eine dieser Farben angeben wollen, verwenden Sie einfach die Auflistung. Den Namensraum `System.Drawing` müssen Sie dazu nicht angeben, weil er wie einige andere Namensräume von ASP.NET automatisch »importiert« wird:

```
txtName.BackColor = Color.Red
```

Andere Farbangaben scheinen wohl nicht möglich zu sein. Der Datentyp `Color` legt die möglichen Farben einfach fest.

Rahmenangaben

Die Eigenschaften, die die Definition eines Rahmens betreffen, sind vom Datentyp `System.Web.UI.WebControls.BorderStyle`, der eine Aufzählung der folgenden Werte ist: `Dashed`, `Dotted`, `Double`, `Groove`, `Inset`, `None`, `Outset`, `NotSet`, `Ridge`, `Solid`). Wenn Sie z. B. den Rahmen eines Label im Programm auf gepunktet setzen wollen, verwenden Sie einen dem folgenden ähnlichen Quellcode:

```
lblDemo.BorderStyle = BorderStyle.Dotted
```

Maßangaben

Maßangaben besitzen bei den ASP.NET-Steuerelementen den Datentyp `System.Web.UI.WebControls.Unit`. Dieser Datentyp ist eine Klasse. Wenn Sie im Programm eine Maßangabe setzen wollen, müssen Sie entweder die Eigenschaft `Empty` (= keine Angabe) oder eine der Methoden dieser Klasse verwenden.

Methode	Beschreibung
`Percentage(`*n*`)`	erzeugt einen Prozent-Wert, dessen Wert im Argument n als `Double` übergeben wird
`Pixel(`*n*`)`	erzeugt einen Pixel-Wert
`Point(`*n*`)`	erzeugt einen Punkt-Wert
`Parse(`*s*`)`	erzeugt einen beliebigen Wert, den Sie im Argument *s* als String übergeben können. Hier können Sie die unter HTML möglichen Einheiten verwenden (`px`, `em`, `cm` etc.).

Tabelle 13.5: Die Methoden der Unit-Klasse zur Erzeugung von Maßangaben

Die Definition einer Maßangabe ist damit vielleicht etwas kompliziert, aber .NET ist eben konsequent objektorientiert. Und Werte wie `"100px"` oder `"2cm"` passen nicht in das OOP-Konzept. Wenn Sie diese Werte aber trotzdem über die in HTML möglichen Einheiten angeben wollen, können Sie ja die `Parse`-Methode verwenden:

```
lblDemo.Width = Unit.Parse("4cm")
lblDemo.Height = Unit.Parse("6mm")
```

Schriftart

Schriftarten werden immer in einem Objekt der Klasse `FontInfo` definiert. Diese Klasse besitzt die Eigenschaften `Bold` (fett), `Italic` (kursiv), `Name` (der Name der Schrift), `Names` (Liste mehrerer Namen), `Overline` (überstrichen), `Size` (Größe), `Strikeout` (durchgestrichen) und `Underline` (unterstrichen).

Die Eigenschaft `Names` ist ein Array aus Schriftarten-Namen. Wenn Sie mehrere Namen angeben wollen, verwenden Sie diese Eigenschaft. Sie müssen dazu ein Array-Objekt der Klasse `String` erzeugen. Beim Erzeugen können Sie dieses direkt initialisieren:

```
txtDemo.Names = New String() {"Verdana", "Times"}
```

Die `Size`-Eigenschaft ist vom Typ `FontUnit`. Hier können Sie einige vordefinierte feste Werte wie `Small`, `Large`, `Smaller` und `Larger` einsetzen:

```
txtDemo.Size = FontUnit.Large
```

Alternativ können Sie die Größe über die `Point`-Methode auch in Punkten definieren:

```
txtDemo.Size = FontUnit.Point(12)
```

Wenn Sie eine andere Einheit verwenden wollen, müssen Sie die `Parse`-Methode verwenden, der Sie einen String mit dem Wert und der Einheit übergeben:

```
txtDemo.Size = FontUnit.Parse("2em")
```

13.3.2 Die Basisklasse WebControl

Alle Webform-Steuerelemente sind von der Basisklasse `WebControl` abgeleitet und besitzen deshalb die Eigenschaften, Methoden und Ereignisse dieser Klasse.

Eigenschaft	Beschreibung
`Attributes`	eine Auflistung aller Attribute des Steuerelements. Über diese Auflistung können Sie neue, benutzerdefinierte Attribute hinzufügen (die Sie vielleicht bei der Auswertung einer Seite wieder auslesen) oder Attributwerte setzen, für die ein Steuerelement keine eigenen Eigenschaften anbietet.

Eigenschaft	Beschreibung
AccessKey	In dieser Eigenschaft können Sie einen Tastatur-Shortcut für das Steuerelement definieren
BackColor	die Hintergrundfarbe
BorderColor	die Farbe des Rahmens
BorderStyle	der Stil des Rahmens
BorderWidth	die Breite des Rahmens
Controls	eine Auflistung aller Steuerelemente, die das Steuerelement als Container beinhaltet
CssClass	In dieser Eigenschaft können Sie einen CSS-Klassennamen definieren um das Steuerelement mit einer CSS-Klasse zu verbinden und damit deren Stile einzusetzen (vgl. Kapitel 5)
Enabled	legt fest, ob das Steuerelement aktiviert ist
EnableViewState	über diese Eigenschaft können Sie festlegen, ob das Steuerelement seinen Status im Viewstate speichert
Font	die Schriftart
ForeColor	die Vordergrundfarbe
Height	die Höhe des Steuerelements
ID	die ID des Steuerelements
Page	eine Referenz zu der Seite, die dieses Steuerelement enthält
Parent	eine Referenz zum Container-Steuerelement, falls das Steuerelement auf einem solchen angelegt ist
Style	eine Auflistung aller Stile
TabIndex	der Tabulator-Index
ToolTip	über diese Eigenschaft können Sie einen Tooltip für das Element einrichten
Visible	definiert, ob das Steuerelement sichtbar ist
Width	die Breite des Steuerelements

Tabelle 13.6: Die wichtigsten Eigenschaften der Basisklasse WebControl

Methode	Beschreibung
DataBind	Diese Methode wird verwendet, wenn ein Steuerelement an eine Datenquelle angebunden wurde, um die Daten anzuzeigen. Datenbindung wird in Kapitel 14 behandelt
FindControl	Über diese Methode können Sie den Behälter des Steuerelements nach einem anderen Steuerelement durchsuchen
HasControls	gibt True zurück, wenn das Steuerelement selbst ein Behälter für andere Steuerelemente ist

Tabelle 13.7: Die Methoden der Basisklasse WebControl

Ereignis	Beschreibung
DataBind	wird aufgerufen, wenn das Steuerelement an eine Datenquelle gebunden wird
PreRender	wird aufgerufen, wenn das Steuerelement beginnt, seinen Inhalt zu rendern

Tabelle 13.8: Die wichtigsten Ereignisse der Klasse WebControl

13.3.3 Die »normalen« Steuerelemente

Das ASP-Label

Das Label ist ein sehr einfaches Steuerelement. Es wird in HTML durch einen Span dargestellt. Die wichtigste Eigenschaft dieses Steuerelements (neben den von WebControl geerbten) ist Text, die den Inhalt verwaltet. Serverseitige Ereignisse besitzt das Label keine (obwohl Sie ja in DHTML clientseitig Ereignisse wie onMouseOver und onMouseOut auswerten können). Die Klasse des Labels ist System.Web.UI.WebControls.Label.

Die ASP-Textbox

Die ASP-Textbox wird verwendet, um einfache Textfelder, Passwortfelder und mehrzeilige Textfelder zu erzeugen. Dieses Verhalten der Textbox steuern Sie über die Eigenschaft TextMode. Möglich sind hier die Werte SingleLine, MultiLine und Password. Das folgende Beispiel erzeugt eine mehrzeilige Textbox:

```
<asp:TextBox id="txtAdress" Height="70px" Width="300px"
 runat="server" TextMode="MultiLine"></asp:TextBox>
```

Die wichtigste Eigenschaft ist wie beim Label die Eigenschaft `Text`, die den dargestellten bzw. eingegebenen Text speichert. Über `MaxLength` bestimmen Sie, wie viele Zeichen der Anwender maximal eingeben darf. Die Eigenschaft `Wrap` bestimmt bei mehrzeiligen Textboxen, ob der Text am rechten Rand automatisch umbricht. Ist `Wrap = False`, erzeugt die Textbox automatisch einen horizontalen Scrollbalken, wenn der Text breiter ist als die sichtbare Fläche der Textbox.

Das wichtigste Ereignis ist `TextChanged`, das aufgerufen wird, wenn der Text vom Anwender geändert wurde.

Die Textbox ist in der Klasse `System.Web.UI.WebControls.Textbox` definiert.

Der ASP-Button

Der Button stellt einen einfachen (Submit)-Schalter dar. Die wichtigste Eigenschaft ist `Text`, die die Beschriftung des Buttons verwaltet. Der folgende Quelltext generiert einen einfachen Button:

```
<asp:Button id="btnOK" runat="server" Text="OK"
 Width="100px" Height="30px"></asp:Button>
```

Wenn Sie auf der Seite Validierungs-Steuerelemente (Seite 385) einsetzen, können Sie über die Eigenschaft `CausesValidation` für einzelne Buttons festlegen, ob eine Betätigung des Schalters eine Validierung auslöst.

Das wichtigste Ereignis des Buttons ist `Click`. Dieses Ereignis wird aufgerufen, wenn der Anwender den Schalter betätigt.

Der LinkButton

Der LinkButton repräsentiert einen Hyperlink, der allerdings nicht als normaler Hyperlink verwendet wird, sondern als eine Art Submit-Button. Deshalb fehlen diesem Steuerelement auch Eigenschaften für die Ziel-URL und den Namen des von ASP.NET zur Darstellung dieses Steuerelements erzeugten Ankers. Die wichtigste Eigenschaft ist `Text`, die den als Link dargestellten Text speichert. Den Submit beim Klicken rea-

lisiert ASP.NET über eine automatisch erzeugte und mit dem `onClick`-Ereignis verbundene JavaScript-Funktion. Das wichtigste Ereignis dieses Steuerelements ist dann auch das `Click`-Ereignis. Der LinkButton ist in der Klasse `System.Web.UI.WebControls.LinkButton` definiert.

Der ImageButton

Statt einem normalen Button können Sie auch einen ImageButton verwenden, den der Benutzer dann zur Ausführung einer Aktion anklicken kann. In der Eigenschaft `ImageUrl` geben Sie die URL für das darzustellende Bild an. `ImageAlign` bestimmt die Ausrichtung des Bildes im Steuerelement. Über die Eigenschaft `AlternateText` bestimmen Sie einen alternativen Text, der von Browsern angezeigt wird, die keine Bilder anzeigen können. `CausesValidation` besitzt dieselbe Bedeutung wie beim Button. Wenn der Anwender den Link betätigt, wird das `Click`-Ereignis ausgelöst.

Der HyperLink

Der HyperLink stellt einen einfachen Hyperlink dar. Die wichtigsten Eigenschaften sind `Text` (die Beschriftung) und `NavigateUrl` (die URL des Ziels). Außer den von `WebControl` geerbten Basisereignissen besitzt dieses Steuerelement keine eigenen.

Da die Eigenschaft `Name` *fehlt, ist die Erzeugung eines Ankers im Dokument, der durch andere Hyperlinks angesprungen werden kann, nicht direkt mit diesem Steuerelement möglich. Verwenden Sie zu diesem Zweck einen selbst erzeugten, normalen HTML-Anker:*

```
<a name="Preilsliste"/>
```

CheckBox und RadioButton

Das `CheckBox`- und das `RadioButton`-Steuerelement sind sich sehr ähnlich. Wie Sie ja bereits wissen, kann der Anwender innerhalb einer Gruppe von `CheckBox`-Elementen auch mehrere auswählen, in einer `RadioButton`-Gruppe allerdings immer nur einen. Die wichtigste Eigenschaft beider Steuerelemente sind `Text` (die Beschriftung) und `Checked` (eingeschaltet). OptionButtons können Sie über die Eigenschaft `GroupName` gruppieren, sodass innerhalb der Gruppe nur einer eingeschaltet werden kann:

```
<asp:RadioButton id="optMusik1" Checked="True"
 GroupName="Musik" runat="server" Text="Hip Hop"/><br>
<asp:RadioButton id="optMusik2" GroupName="Musik"
 runat="server" Text="R & B"/><br>
<asp:RadioButton id="optMusik3" GroupName="Musik"
 runat="server" Text="Drum & Bass"/><br>
<asp:RadioButton id="optMusik4" GroupName="Musik"
 runat="server" Text="Trance"/><br>
```

Bei der Auswertung fragen Sie die Eigenschaft `Checked` ab. Bei OptionButtons haben Sie mit einem kleinen Select-Case-Trick dabei recht wenig Arbeit:

```
Select Case True
   Case optMusik1.Checked
      txtResult.Text = "Sie hören gerne Hip Hop"
   Case optMusik2.Checked
      txtResult.Text = "Sie hören gerne R & B"
   Case optMusik3.Checked
      txtResult.Text = "Sie hören gerne Drum & Bass"
   Case optMusik4.Checked
      txtResult.Text = "Sie hören gerne Alternative"
End Select
```

Beachten Sie, dass Sie Gruppen von OptionButtons oder CheckBoxen viel einfacher über das `OptionButtonList`*- oder das* `CheckBoxList`*-Steuerelement erzeugen und auswerten können (ab Seite 381).*

Image

Das `Image`-Steuerelement stellt einfach ein Bild im HTML-Dokument dar (in einem `img`-Tag). Wenn Sie im Programm auf die Eigenschaften diese Bildes zugreifen wollen (z. B. um die URL des Bildes dynamisch zu verändern), können Sie dieses Steuerelement dazu einsetzen. Die wichtigsten Eigenschaften sind `ImageUrl` (URL der Bilddatei) und `ImageAlign` (Ausrichtung des Bildes).

```
<asp:Image id="imgLogo" runat="server" Width="100px"
 Height="100px" ImageUrl="logo.gif"
 ImageAlign="Middle"></asp:Image>
```

Serverseitige Ereignisse besitzt dieses Steuerelement nicht.

Panel

Das Panel-Steuerelement ist ein `div`-Element, für das Sie die Größe, den Stil und ein Hintergrund-Bild definieren können. Da Sie (natürlich) neben Text auch andere Steuerelemente auf (bzw. in) einem Panel platzieren können, können Sie das Panel als Layout-Element einer Seite einsetzen. Es eignet sich auch gut dazu, eine Gruppe von Steuerelementen oder Text dynamisch im Programm sichtbar oder unsichtbar zu schalten. Wenn Sie die `Visible`-Eigenschaft eines Panel auf `False` setzen, sind auch alle enthaltenen Steuerelemente unsichtbar. In einer ASP.NET-Seite, deren Steuerelemente nicht absolut oder relativ positioniert sind, benötigt ein unsichtbares Panel keinen Platz auf der Seite, sodass kein Leerraum entsteht.

Neben den von `WebControl` geerbten Eigenschaften (die ja hier für die Definition des Aussehens wichtig sind) besitzt das `Panel` die wichtigen Eigenschaften `HorizontalAlign` (horizontale Ausrichtung des Inhalts) und `Wrap` (automatischer Umbruch des Inhalts am rechten Rand).

Serverseitige Ereignisse besitzt dieses Steuerelement nicht.

Calendar

Das Calendar-Steuerelement ist ein mächtiges Steuerelement zur Darstellung eines Kalenders.

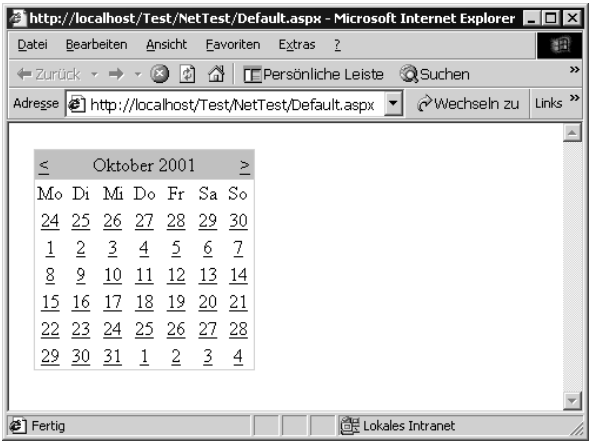


Bild 13.1: Das Calendar-Steuerelement im Einsatz

Über verschiedene Eigenschaften können Sie die optische Gestaltung des Kalenders beeinflussen. Das Steuerelement reagiert automatisch auf eine Auswahl des Benutzers dadurch, dass zum einen die Ansicht des Kalenders auf die Auswahl umgestellt und zum anderen das Ereignis `SelectionChanged` aufgerufen wird. In diesem Ereignis können Sie auf den Wechsel des ausgewählten Datums reagieren.

Je nach Browser soll das Steuerelement sogar JavaScript-Code erzeugen, der bei einem Wechsel des aktiven Datums dafür sorgt, dass dieser auf dem Client ausgeführt werden kann. Bei meinen Tests hat das aber (natürlich) wie so oft nicht funktioniert, das Steuerelement führte bei jedem Datumswechsel einen kompletten Server-Roundtrip aus.

Das Calendar-Steuerelement besitzt eine sehr große Anzahl Eigenschaften. Viele davon betreffen das Aussehen. So finden Sie in der (Objekt-)Eigenschaft `DayStyle` z. B. Eigenschaften zur Einstellung des Stils des Datumsbereichs des Kalenders. Ich kann diese Eigenschaften hier nicht alle beschreiben, probieren Sie dieses Steuerelement einfach aus.

Die wichtigste Eigenschaft ist `SelectedDate`. Diese Eigenschaft gibt das ausgewählte Datum zurück. Das wichtigste Ereignis ist `SelectionChanged`, das aufgerufen wird, wenn der Anwender die Auswahl (also das Datum) ändert.

Das folgende Beispiel reagiert auf eine Änderung des Datums mit dem Schreiben des Datums in eine Textbox:

```
Private Sub cal_SelectionChanged( _
   ByVal sender As System.Object, _
   ByVal e As System.EventArgs) _
   Handles cal.SelectionChanged
   txtDate.Text = cal.SelectedDate.ToLongDateString()
End Sub
```

AdRotator

Das `AdRotator`-Steuerelement können Sie verwenden, um automatisch wechselnde Bilder (u. a. für Werbung) in eine Seite einzublenden. Die anzuzeigenden Datei-URLs werden in einer XML-Datei

definiert, die in der Eigenschaft `AdvertisementFile` angegeben wird. Diese Datei sieht zum Beispiel so aus wie im folgenden Listing:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Advertisements>
  <Ad>
    <ImageUrl>nitty-gritty.gif</ImageUrl>
    <NavigateUrl>http://www.nitty-gritty.de</NavigateUrl>
    <AlternateText>Nitty-Gritty</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>Topic1</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>aspheutetitle.gif</ImageUrl>
    <NavigateUrl>http://www.aspheute.com</NavigateUrl>
    <AlternateText>ASP Heute</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>Topic2</Keyword>
  </Ad>
</Advertisements>
```

Die Elemente der XML-Datei sind fast alle meist erklärend. Das Element `Impression` steuert, wie häufig das Bild in Relation zu anderen Bildern angezeigt wird. Sind die Werte aller Ads gleich groß, werden diese gleich häufig angezeigt. Im Element `Keyword` geben Sie eine Kategorie an. Über die Eigenschaft `KeywordFilter` können Sie später eine dieser Kategorien angeben, um nur die Ads einer bestimmten Kategorie anzuzeigen. In der Eigenschaft Target definieren Sie, wo die Ziel-URL geöffnet wird (z. B. "_blank" für ein neues Fenster, "_self" für das aktuelle Fenster):

```
<asp:AdRotator id="adrLinks" AdvertisementFile="ads.xml"
 Target="_self" runat="server"/>
```

Beachten Sie, dass das jeweils andere Bild immer nur angezeigt wird, nachdem die Seite komplett neu angefordert wurde.

Table

Das `Table`-Steuerelement erzeugt eine HTML-Tabelle. Sie können dieses Steuerelement nutzen, um Tabellen (in Visual Studio) im Entwurf

recht einfach (aber komplex) zu gestalten oder um Tabellen dynamisch im Quellcode zu erzeugen. Die Eigenschaften des `Table`-Steuerelements entsprechen meist den Attributen des HTML-`table`-Tag, weswegen ich hier auf eine Beschreibung verzichte. Die Eigenschaft `GridLines` steuert, wo die Tabelle Gitternetzlinien ausgibt (`Horizontal`, `Vertical`, `Both`).

Das folgende Beispiel erzeugt eine leere Basistabelle mit Rahmen, Gitternetzlinien, CellPadding und CellSpacing:

```
<asp:Table id="tblDemo" runat="server"
 Height="33px" Width="407px" BorderWidth="1px"
 CellPadding="2" CellSpacing="2" GridLines="Both"/>
```

Eine wichtige Eigenschaft ist `Rows`, eine Auflistung von `TableRow`-Elementen, deren wichtigste Eigenschaft (neben den normalen Eigenschaften einer Tabellenzeile) `Cells` ist, eine Auflistung von `TableCell`-Objekten.

Da Tabellen häufig im Quellcode dynamisch erzeugt werden, folgt hier ein Beispiel zur Erzeugung der Zeilen und Spalten der Tabelle:

```
' Tabelle dynamisch füllen
Dim intRow As Integer, intCol As Integer
Dim objRow As TableRow, objCell As TableCell
For intRow = 1 To 3
    ' Neue Zeile erzeugen
    objRow = New TableRow()
    For intCol = 1 To 4
        ' Neue Zelle erzeugen
        objCell = New TableCell()
        objCell.Text = "Row " & intRow & ", Col " & intCol
        ' Zelle der Zeile hinzufügen
        objRow.Cells.Add(objCell)
    Next
    ' Zeile der Tabelle hinzufügen
    tblDemo.Rows.Add(objRow)
Next
End Sub
```

13.3.4 Listen-Steuerelemente

ASP.NET definiert einige Steuerelemente, die eine Liste von Einträgen speichern. Dazu gehören das `DropDownList`- (einzeilige Liste), das `ListBox`- (mehrzeilige Liste), das `CheckBoxList`- (Liste von Checkboxen) und das `RadioButtonList`-Steuerelement (Liste von RadioButtons).

Die Klassen dieser Steuerelemente sind von der Klasse `ListControl` abgeleitet.

Die ListControl-Basisklasse

Die `ListControl`-Basisklasse, die selbst von `WebControl` abgeleitet ist, definiert das Verhalten der Liste der Listen-Steuerelemente.

Eigenschaft	Beschreibung
`AutoPostBack`	bestimmt, ob eine Änderung des aktuellen Eintrags durch den Benutzer zu einem Submit und damit zur Ausführung des `SelectedIndexChanged`-Ereignisses führt.
`Items`	eine Auflistung der gespeicherten Elemente. Elemente sind vom Typ `ListItem`.
`SelectedIndex`	verwaltet bei Einfachauswahl-Listen den Index des selektierten Eintrags
`SelectedItem`	gibt bei Einfachauswahl-Listen eine Referenz auf den selektierten Eintrag zurück

Tabelle 13.9: Die wichtigsten Eigenschaften der ListControl-Klasse (ohne die für Datenbindung verwendeten)

Ereignis	Beschreibung
`OnSelected-ItemChanged`	wird aufgerufen, wenn der Benutzer einen anderen Eintrag selektiert hat

Tabelle 13.10: Die wichtigsten Ereignisse der ListControl-Klasse (ohne die für Datenbindung verwendeten)

Hinzufügen von Elementen

Bei allen Listen-Steuerelementen füllen Sie die Liste über die `Add`-Methode der `Items`-Auflistung. Diese Auflistung speichert Objekte der Klasse ListItem.

Wenn Sie die Liste im Quellcode füllen, können Sie wahlweise einzelne ListItem-Objekte erzeugen und der `Add`-Methode übergeben oder einfach nur Strings übergeben. Das folgende Beispiel füllt ein `DropDownList`-Steuerelement im Load-Ereignis der Seite:

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
   If IsPostBack = False Then
      With lstLieblingsPerson.Items
         .Add("Zaphod")
         .Add("Trillian")
         .Add("Ford")
         .Add("Arthur")
      End With
   End If
End Sub
```

Handelt es sich um ein `CheckBoxList`- oder `RadioButtonList`-Steuerelement, das Sie dynamisch im Programm füllen, müssen Sie ein wenig aufpassen, wenn die Elemente der Seite absolut positioniert werden. Diese Steuerelemente werden natürlich automatisch vergrößert, wenn sie gefüllt werden, und überlagern dann u. U. andere Elemente der Seite.

Bei einem `RadioButtonList`-Steuerelement ist nach dem Füllen keiner der Schalter eingeschaltet. Für die Praxis ist dies meist nicht sinnvoll, da Sie normalerweise erwarten, dass eine Option gewählt wird. Schalten Sie also eine Option ein, indem Sie `SelectedIndex` auf deren Index setzen:

```
' Voreinstellung des ersten Eintrags
lstMusik.SelectedIndex = 0
```

Löschen der Liste

Mit der `Clear`-Methode der `Items`-Ausflistung können Sie die Liste im Programm löschen:

```
lstLieblingsPerson.Items.Clear()
```

Auswerten von Einfachauswahl-Listen

Das `DropDownList`-, das `RadioButtonList`- und das `ListBox`-Steuerelement mit `SelectioMode = Single` erlauben nur eine Einfachauswahl. Wenn Sie diese auswerten wollen, ermitteln Sie einfach den Index des selektierten Elements. Diesen Index können Sie dann verwenden, um den selektierten Eintrag auszulesen:

```
Dim i As Integer
i = lstLieblingsPerson.SelectedIndex
If i > -1 Then
   lstResult.Items.Add("Ihre Lieblingsperson ist " & _
      "die Person Nummer " & _i & ": " & _
      lstLieblingsPerson.Items(i).Value)
Else
   lstResult.Items.Add("Sie haben keine Lieblingsperson")
End If
```

Den selektierten Eintrag erhalten Sie aber auch über die Eigenschaft `SelectedItem`:

```
   lstResult.Items.Add("Ihre Lieblingsperson ist " & _
      "die Person Nummer " & _i & ": " & _
      lstLieblingsPerson.SelectedItem.Value)
```

Auswerten von Mehrfachauswahl-Listen

Zu den Mehrfachauswahl-Listen gehören das `CheckBoxList`- und das `Listbox`-Steuerelement mit `SelectionMode = Multiple`. Diese Listen müssen Sie etwas anders auswerten, weil ja mehrere Auswahlen möglich sind. Gehen Sie dazu die Liste sequenziell durch und überprüfen Sie über die `Selected`-Eigenschaft des `ListItem`-Elements, ob dieses selektiert ist:

```
For i = 0 To lstAutos.Items.Count - 1
   If lstAutos.Items.Item(i).Selected Then
      lstResult.Items.Add("Sie fahren gerne einen " & _
         lstAutos.Items(i).Value)
   End If
Next
```

13.3.5 Spezielle Steuerelemente

Literal und PlaceHolder

Das `Literal`-Steuerelement ist nichts weiter als ein serverseitiger Container für reinen HTML-Text. Die `Text`-Eigenschaft ist dann (neben `EnableViewState`) auch die einzige wichtige Eigenschaft dieses Steuerelements. Nutzen können Sie das Literal-Steuerelement, wenn Sie reinen Text oder selbst erzeugten HTML-Quelltext an einer definierten Stelle in der Seite ausgeben wollen. Naturgemäß können Sie dieses Steuerelement auch nicht positionieren.

Das `PlaceHolder`-Steuerelement dient als Platzhalter (wofür auch sonst, das sagt ja schon der Name ...). Wenn Sie Steuerelemente dynamisch erzeugen und keine absolute Positionierung (die ja die Stile `Left` und `Top` einsetzt) verwenden wollen, können Sie den Platzhalter auf der Seite platzieren und die Steuerelemente innerhalb des Platzhalters erzeugen. Weitere Informationen dazu finden Sie ab Seite 393.

Xml

Das `Xml`-Steuerelement können Sie verwenden, um den Inhalt eines XML-Dokuments anzuzeigen oder um ein Dokument über ein XSL- oder XSLT-Stylesheet auf dem Server zu transformieren[1]. Die wichtigsten Eigenschaften dieses Steuerelements, das übrigens nicht von `WebControl`, sondern direkt von der `Control`-Klasse abgeleitet ist, stellt Tabelle 13.11 dar.

1. Prinzipiell speichert ein XML-Dokument strukturierte Daten, ähnlich einer Datenbank. Diese Daten können über Stylesheets so formatiert werden, dass sie in einer lesbaren und optisch ansprechenden Form dargestellt werden.

Eigenschaft	Beschreibung
`Document`	eine Referenz zu einem `XmlDocument`-Objekt, das das XML-Dokument repräsentiert
`Document-Content`	diese Eigenschaft gibt den Inhalt des XML-Dokuments als String zurück
`Document-Source`	der physische oder virtuelle Pfad zum Dokument
`Transform`	eine Referenz zu einem `XmlTransform`-Objekt, das die Daten des XSL- bzw. XSLT-Stylesheets verwaltet, das für die Transformierung verwendet wird
`Transform-FromSource`	eine Pfadangabe zu einem XSL- oder XSLT-Dokument, das zur Transformierung verwendet werden soll

Tabelle 13.11: Die wichtigsten Eigenschaften des Xml-Steuerelements

13.3.6 Die Validierungs-Steuerelemente

Die Validierungs-Steuerelemente sind in der Praxis enorm hilfreich. Die Überprüfung von Benutzereingaben ist eine sehr wichtige Aufgabe bei der Programmierung. Ohne die Validierungs-Steuerelemente bleiben Ihnen dazu nur zwei Möglichkeiten: Das serverseitige Überprüfen in ASP und das clientseitige mit JavaScript. Beides erfordert einen hohen Programmieraufwand. Die Validierungs-Steuerelemente nehmen Ihnen diese Arbeit ab.

Das Prinzip dieser Steuerelemente ist einfach: Sie sehen aus wie einfache Label, werden auf einem Webformular an geeigneter Stelle platziert und mit einem Steuerelement verknüpft, das sie überwachen sollen. Abhängig von der Art der Validierung definieren Sie eine Regel. Der `RequiredFieldValidator` verlangt z. B. nur, dass im verbundenen Steuerelement eine Eingabe erfolgt. Bestätigt der Anwender das Formular, überprüfen die Validierungs-Steuerelemente daraufhin, ob die Regeln eventuell verletzt wurden, und brechen die Aktion in diesem Fall ab. Zusätzlich zeigen sie noch einen Fehlertext an.

Das folgende Beispiel zeigt, wie einfach das Ganze ist. Ein `RequiredFieldValidator`-Steuerelement ist mit einer Textbox verbunden und überprüft, ob dort Eingaben erfolgt sind:

```
<form id="Form1" method="post" runat="server">
 Name: <asp:TextBox ID="txtName"
  Runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator id="val1" runat="server"
  ErrorMessage="Geben Sie bitte Ihren Namen ein"
  ControlToValidate="txtName">
 </asp:RequiredFieldValidator>
 <p>
 <asp:Button ID="btnOK" Text="OK" Width="100px"
  Runat="server"></asp:Button> 
</form>
```

Wenn die Seite ausgeführt wird, ist das Validierungs-Steuerelement zunächst unsichtbar (Abbildung 13.2).

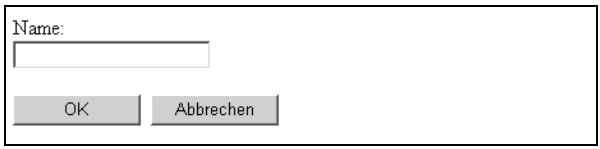


Bild 13.2: Ein Webform mit Validierungs-Steuerelement vor dem Absenden des Formulars

Verletzt der Anwender die Regeln und versucht, das Formular abzusenden, werden die betroffenen Validierungs-Steuerelemente sichtbar (Abbildung 13.3).

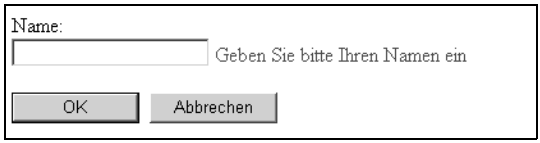


Bild 13.3: Ein Validierungs-Steuerelement ist sichtbar geworden, weil der Anwender die Regeln verletzt hat

Alternativ können Sie alle Fehlermeldungen gemeinsam in einem `ValidationSummary`-Steuerelement ausgeben lassen. Dazu ist zunächst nichts weiter notwendig, als dieses Steuerelement auf der aspx-Seite zu platzieren und die Ausgabe der Fehlermeldung für die einzelnen Validierungs-Steuerelemente zu unterdrücken:

```
<form id="Form1" method="post" runat="server">
 Name:<br>
 <asp:TextBox ID="txtName" Runat="server"/>
 <asp:RequiredFieldValidator id="valName" runat="server"
  ErrorMessage="Der Name muss eingegeben werden"
  Display="None"
  ControlToValidate="txtName"/>
 <p/>

 Ihr Alter:<br>
 <asp:TextBox ID="txtAlter" Runat="server"/>
 <asp:RequiredFieldValidator id="valAlter1" runat="server"
  ErrorMessage="Das Alter muss eingegeben werden"
  Display="None"
  ControlToValidate="txtAlter"/>
 <asp:RangeValidator id="valAlter2" runat="server"
  ErrorMessage="Das Alter muss zwischen 0 und 150 liegen"
  MinimumValue="0" MaximumValue="150" Type="Integer"
  Display="None"
  ControlToValidate="txtAlter"/>
 <p/>

 <asp:ValidationSummary id="valSummary" runat="server"
  Height="80px" Width="260px" ShowMessageBox="false"
  ShowSummary="true"
  HeaderText="Die folgenden Fehler sind aufgetreten"
  DisplayMode="List"/>

 <asp:Button ID="btnOK" Text="OK" Width="100px"
  Runat="server"></asp:Button> 
</form>
```

Wie Sie dem Beispiel entnehmen können, können Sie auch mehrere Validierungs-Steuerelemente für ein Steuerelement anlegen. Das Alter wird z. B. auf Vorhandensein und auf einen Bereich überprüft.

Die Anzeige der Fehlermeldungen der einzelnen Validierungs-Steuerelemente ist über `DisplayMode="None"` ausgeschaltet. Werden nun Regeln verletzt, zeigt das `ValidationSummary` die einzelnen Fehlermeldungen an.

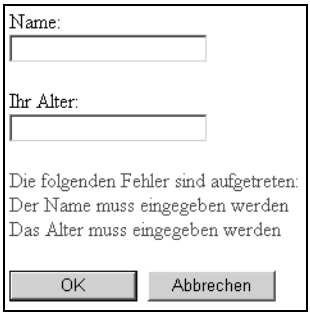


Bild 13.4: Anzeige von Fehlermeldungen durch das ValidationSummary-Steuerelement

Die Überprüfung der Regeln erfolgt für alle Validierungs-Steuerelemente außer dem `CustomValidator` im Browser, sofern Sie nicht die Eigenschaft `EnableClientScript` ausschalten und wenn der Browser in der Lage ist, JavaScript-Code auszuführen. ASP.NET erzeugt (für den Internet Explorer) ziemlich komplexen Code, den ich hier gar nicht darstellen will (erstaunlicherweise ist der Code für den Netscape 4.7 wesentlich einfacher). Schauen Sie sich einfach den Quelltext der erzeugten HTML-Seite an.

Die Eigenschaft CausesValidation

Alle Steuerelemente, über die der Anwender ein Formular absenden kann, besitzen die Eigenschaft `CausesValidation`. Standardmäßig steht der Wert dieser Eigenschaft auf `True`. Wenn Sie den Wert auf `False` setzen, führt das dazu, dass bei der Betätigung dieses Steuerelements keine Validierung erfolgt. Dieses Verhalten ist sehr hilfreich, wenn Sie Steuerelemente auf der Seite platzieren, über die der Anwender die Aktion abbrechen kann. Bei einem Abbruch soll ja eigentlich keine Validierung erfolgen. Das folgende Code-Fragment stellt den unteren Teil des Beispiels dar, dieses Mal aber mit einem Abbrechen-Schalter:

```
<asp:Button ID="btnOK" Text="OK" Width="100px"
 Runat="server"></asp:Button> 
<asp:Button ID="btnEsc" Text="Abbrechen" Width="100px"
 CausesValidation="False" Runat="server"></asp:Button>
```

Die Klasse BaseValidator

Alle Validierungs-Steuerelemente sind von der Basisklasse `BaseValidator` abgeleitet und erben deswegen die in Tabelle 13.2 zusammengefassten wichtigen Eigenschaften. `BaseValidator` selbst erbt alle Elemente der Klasse `WebControl` (Seite 371).

Eigenschaft	Bedeutung
`ControlToValidate`	Hier geben Sie die ID des Steuerelements an, das überprüft werden soll
`Display`	legt das Verhalten der Fehlermeldung fest. Möglich sind die Werte `None` (keine Fehlermeldung), `Static` und `Dynamic`. Wenn Sie Display auf `Dynamic` setzen, wird das Element zunächst in der Breite 0 angelegt. Wird der Fehler ausgegeben, wird das Element vergrößert, was dazu führt, dass rechts davon angelegte Elemente verschoben werden, wenn Sie keine absolute Positionierung verwenden.
`EnableClientScript`	In dieser Eigenschaft können Sie festlegen, ob eine clientseitige Validierung erfolgen soll, falls dies möglich ist
`Enabled`	legt fest, ob das Steuerelement aktiviert ist, also seiner Aufgabe nachgeht
`ErrorMessage`	die Fehlermeldung
`IsValid`	Über diese Eigenschaft können Sie herausfinden, ob das verbundene Steuerelement valide Daten enthält

Tabelle 13.12: Die wichtigen Eigenschaften der Klasse BaseValidator

Über die einzige wichtige Methode `Validate` können Sie eine Validierung im Programm initiieren.

RequiredFieldValidator

Ein RequiredFieldValidator-Steuerelement überprüft, ob in dem verbundenen Steuerelement Eingaben erfolgt sind. Die wichtigsten Eigenschaften sind die der `BaseValidator`-Klasse.

CompareValidator

Ein `CompareValidator`-Steuerelement überprüft, ob zwei Steuerelemente denselben Wert besitzen. Solche Validierungen benötigen Sie z. B., wenn der Benutzer ein Passwort eingeben und noch einmal wiederholen soll. Neben den Eigenschaften der `BaseValidator`-Klasse ist die Eigenschaft `ControlToCompare` wichtig. Hier geben Sie an, mit welchem Steuerelement verglichen werden soll.

RangeValidator

Ein `RangeValidator`-Steuerelement überprüft, ob eine Eingabe in einem angegebenen Wertebereich liegt. Den minimalen und maximalen Wert geben Sie in den Eigenschaften `MinimumValue` und `MaximumValue` an. In der `Type`-Eigenschaft legen Sie den Datentyp fest: `String`, `Integer`, `Double`, `Date` oder `Currency`.

RegularExpressionValidator

Das RegularExpressionValidator-Steuerelement ermöglicht die Verwendung eines komplexen regulären Ausdrucks[1] für die Überprüfung. Diesen geben Sie in der Eigenschaft `ValidationExpression` an. Komplexe Ausdrücke werden in diesem Buch nicht behandelt. Das folgende Beispiel überprüft eine Eingabe darauf, ob es sich um eine gültige Postleitzahl handelt:

```
Postleitzahl:<br>
<asp:TextBox ID="txtPlz" Runat="server"/>
<!-- RequiredFieldValidator -->
<asp:RequiredFieldValidator id="valPlz1" runat="server"
 ErrorMessage="Das Plz muss eingegeben werden"
 Display="None"
 ControlToValidate="txtPlz"/>
<!-- RegularExpressionValidator -->
<asp:RegularExpressionValidator id="valPlz2"
```

1. Reguläre Ausdrücke ermöglichen die flexible Überprüfung von Daten auf bestimmte Regeln. Der Ausdruck »(D-)?\d{5}« überprüft z. B., ob es sich um eine in Deutschland gültige Postleitzahl handelt, der Ausdruck »\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*« überprüft, ob eine angegebene Mailadresse gültig ist.

```
 runat="server"
 ErrorMessage="Die Postleitzahl ist ungültig"
 ValidationExpression="(D-)?\d{5}"
 Display="None"
 ControlToValidate="txtPlz"/>
```

Beachten Sie, dass wieder zusätzlich ein `RequiredFieldValidator` eingerichtet wurde, um eine leere Eingabe abzufangen.

CustomValidator

Das `CustomValidator`-Steuerelement erlaubt eigene Validierungen. Sie können dazu eine clientseitige eigene JavaScript-Funktion einsetzen, deren Namen Sie in der Eigenschaft `ClientValidationFunction` angeben. Alternativ können Sie im serverseitigen Ereignis `ServerValidate` validieren, dann ist allerdings natürlich ein Server-Roundtrip notwendig.

Der folgende Quelltext zeigt, wie eine clientseitige Validierung funktioniert. Die Funktion `CheckPrimeNumber` überprüft, ob die Eingabe eine Primzahl ist. Diese Funktion bekommt zwei Argumente übergeben, eine Referenz auf das Steuerelement und eine Referenz auf ein Objekt der Klasse `EventArgs`. Diese Klasse besitzt die Eigenschaften `Value` (die Eingabe) und `IsValid`. Über `IsValid` definieren Sie, ob die Eingabe gültig ist:

```
<script language="JavaScript">
 /* Validierungsfunktion für den CustomValidator */
 function CheckPrimeNumber(source, args)
 {
   var i;
   args.IsValid = false;
   if (isNaN(args.Value)) return;
   for (i=2; i<args.Value; i++)
      if (args.Value % i == 0) return;
   args.IsValid = true;
   return;
 }
</script>
```

Achten Sie darauf, dass JavaScript Groß- und Kleinschreibung unterscheidet. Abweichend von der üblichen Vorgehensweise unter JavaScript müssen Sie die Eigenschaften `Value` *und* `IsValid` *mit großen Anfangsbuchstaben schreiben, ansonsten erzeugt die JavaScript-Engine einfach neue Variablen und Ihr Quellcode funktioniert nicht.*

Diese Funktion geben Sie dann als `ClientValidationFunction` an:

```
Geben Sie eine möglichst große Primzahl an:<br>
<asp:TextBox ID="txtPrimzahl" Runat="server"/>
<!--CustomValidator -->
<asp:CustomValidator id="valPrimzahl"
 runat="server"
 ErrorMessage="Das ist keine Primzahl"
 ClientValidationFunction="CheckPrimeNumber"
 Display="None"
 ControlToValidate="txtPrimzahl"/>
```

Wenn Sie serverseitig validieren wollen, müssen Sie lediglich das `ServerValidate`-Ereignis definieren. Die Argumente sind dieselben wie bei der clientseitigen JavaScript-Funktion:

```
Private Sub valPrimzahl_ServerValidate( _
   ByVal source As System.Object, ByVal args As _
   System.Web.UI.WebControls.ServerValidateEventArgs)_
   Handles valPrimzahl.ServerValidate
 Dim i As Integer
 args.IsValid = False
 If Not IsNumeric(args.Value) Then Return
 For i = 2 To CInt(args.Value) - 1
   If (CInt(args.Value) Mod i) = 0 Then Return
 Next
 args.IsValid = True
End Sub
```

ValidationSummary

Das `ValidationSummary`-Steuerelement setzen Sie ein, um alle Fehlermeldungen gemeinsam auszugeben. Dazu reicht ein einfaches Plat-

zieren dieses Steuerelements auf dem Webformular aus. Sie können das Verhalten über einige Eigenschaften beeinflussen. Die Eigenschaft `HeaderText` verwaltet eine Zeichenkette, die als Überschrift ausgegeben wird. In der Eigenschaft `DisplayMode` können Sie die Art der Liste einstellen. Möglich sind die Werte `List` (einfache Liste), `Bullet` (Aufzählung) und `SingleParagraph` (alle Texte hintereinander). Interessant ist auch die Eigenschaft `ShowMessageBox`. Wenn Sie diese Eigenschaft auf `True` setzen, wird clientseitig zusätzlich zu der Fehlermeldung eine MessageBox mit den Fehlertexten ausgegeben. Wenn Sie nur die MessageBox verwenden wollen, können Sie die Eigenschaft `ShowSummary` auch auf False einstellen.

13.4 Dynamisches Erzeugen von Steuerelementen

Der dynamische Aufbau von HTML-Seiten ist ein wichtiger Aspekt der ASP(.NET)-Programmierung. Unter ASP werden dazu einfach die entsprechenden HTML-Tags dynamisch erzeugt, z. B. um die Daten einer Datenbanktabelle in einer HTML-Tabelle darzustellen. Natürlich können Sie das auch unter ASP.NET so machen. Sie haben aber auch die Möglichkeit, Steuerelemente dynamisch zu erzeugen.

Beachten Sie, dass einige Steuerelemente, die an Datenquellen angebunden werden können, die dynamische Anzeige dieser Daten erheblich vereinfachen. Besonders das (komplexe) `Repeater`*-Steuerelement bietet viele Möglichkeiten, ohne dass Sie Steuerelemente selbst dynamisch erzeugen müssen.*

Unter ASP.NET ist es sehr einfach, Steuerelemente dynamisch zu erzeugen. Sie erzeugen einfach ein Objekt der Klasse des Steuerelements, setzen dessen Eigenschaften, verbinden das Objekt mit Ereignisprozeduren und fügen es dem `Page`-Objekt oder einem Container-Steuerelement wie einem `Panel` hinzu.

Das folgende Beispiel erzeugt drei Textboxen, wenn die Seite geladen wird:

```
Private Sub Page_Load()
 Dim i As Integer, txt As WebControls.TextBox
 For i = 1 To 3
```

```
    ' Neue Textbox erzeugen
    txt = New WebControls.TextBox()
    ' Eigenschaften setzen
    txt.Style.Add("position", "absolute")
    txt.Style.Add("left", "10px")
    txt.Style.Add("top", 30 + (i * 25) & "px")
    txt.Width = Unit.Pixel(100)
    txt.Height = Unit.Pixel(20)
    txt.BackColor = System.Drawing.Color.Yellow
    ' Textbox der Seite hinzufügen
    Me.Controls.Add(txt)
 Next
End Sub
```

Beachten Sie, dass einige Eigenschaften wie die Position über Stile gesetzt werden müssen. Sie können natürlich auch darauf verzichten und die Steuerelemente ohne Positionierung hinter- und untereinander anlegen.

14 ASP.NET-Datenbindung und -Datenbankzugriff

Einige Steuerelemente können Sie mit nur wenigen Zeilen Quellcode an eine Datenquelle binden. Diese Steuerelemente zeigen die Daten der Quelle dann automatisch an. Mit dem `DataGrid`-Steuerelement ist es so kein Problem, eine Datenquelle (z. B. ein `DataSet` mit den Daten einer Datenbankabfrage) tabellenförmig anzuzeigen. Die Datenbindung arbeitet mit einfachen Datenquellen wie Auflistungen oder Arrays, aber auch mit den Datenbank-Objekten. Dieses Kapitel beschreibt zunächst die Prinzipien der Datenbindung und geht danach auf das Abfragen und Bearbeiten von externen Datenquellen mit ADO.NET ein.

14.1 Datenbindung

Datenbindung existiert in zwei Varianten. Die Einfachwert-Datenbindung bindet eine Variable, eine Eigenschaft oder eine Funktion an ein Steuerelement. Das ist ziemlich identisch mit der Verwendung von Inline-ASP mitten im HTML-Quelltext (`<%=`*Ausdruck*`%>`). Die andere Variante ist die eigentliche Datenbindung: Die Bindung von Datenquellen mit mehreren Zeilen und Spalten an passende Steuerelemente.

Bei beiden Varianten sind zwei Dinge wichtig: Das Auslesen und Darstellen der Daten erfolgt auf dem Server. Der Client erhält lediglich das Ergebnis. Der zweite wichtige Punkt ist, dass die Daten immer nur gelesen werden. Sofern der Benutzer die Daten verändert, werden diese nicht implizit in die Datenquelle zurückgeschrieben (wie es z. B. unter Visual Basic 6 der Fall war).

Die Datenbindung ist sehr mächtig. Ich kann in diesem Buch nur einen kleinen (aber wichtigen) Teil davon zeigen. Besonders die Steuerelemente `DataGrid`, `DataList` und `DataRepeater` bieten noch wesentlich mehr Möglichkeiten, als ich hier zeigen kann. Weil alleine die Anzahl der Eigenschaften und Ereignisse so groß ist, dass deren Beschreibung den Rahmen dieses Buchs sprengen würde, stelle ich die wichtigen Datenbindungs-Features einfach an Beispielen vor und verzichte auf eine Beschreibung der Eigenschaften und Ereignisse der speziellen Steuerelemente.

14.1.1 Einfachwert-Datenbindung

Mit der Einfachwert-Datenbindung können Sie Steuerelemente an beliebige Ausdrücke binden. Sie verwenden dazu eine Syntax, die stark an Inline-ASP erinnert:

```
<%# Ausdruck%>
```

Der Unterschied zu Inline-ASP ist, dass Sie diese Syntax auch in ASP-Steuerelement-Deklarationen einsetzen können:

```
<asp:Textbox id="txtDate"
 Text="<%# System.DateTime.Now.ToShortDateString()%>"
 runat="server" />
```

Weil der Ausdruck ein beliebiger Ausdruck sein kann, können Sie natürlich auch Variablen, Eigenschaften, Funktionen und Methoden einsetzen. Das Beispiel oben ruft ja bereits eine Funktion auf.

Damit die Datenbindung ausgeführt wird, müssen Sie die `DataBind`-Methode des Steuerelements oder der gesamten Seite aufrufen:

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
   Page.DataBind()
End Sub
```

Wenn Sie die `DataBind`-Methode der Seite aufrufen, werden alle gebundenen Steuerelemente aufgefordert, ihren Inhalt aus der Datenquelle auszulesen. Für Steuerelemente, die speziell für Datenbin-

dung entwickelt wurden (wie z. B. die `Listbox` und das `DataGrid`), können Sie auch deren separate `DataBind`-Methode aufrufen, wenn nur diese Steuerelemente aktualisiert werden sollen. Einfache Steuerelemente wie die HTML-Steuerelemente besitzen diese Methode aber nicht.

Die Einfachwert-Datenbindung können Sie für beliebige Eigenschaften von HTML- und ASP-Steuerelementen einsetzen. HTML-Steuerelemente müssen dazu noch nicht einmal serverseitig ausgeführt werden:

```
<input type="text"
 value="<%# System.DateTime.Now.ToShortDateString()%>">
```

Das folgende Beispiel verwendet diese Technik, um die URL eines `Image`-Steuerelements bei jeder Anforderung der Seite zufällig zu ermitteln:

```
<html><head>
<script language="VB" runat="Server">
 Private Function GetImageUrl() As String
  Dim i As Integer, seed As Integer
  seed = System.Convert.ToInt32( _
     DateTime.Now.Ticks And &H7FFFFFFF)
  Dim rnd As Random = New Random(seed)
  i = rnd.Next(1, 5)
  return "cereal" & i & ".gif"
End Function
</script>
</head>

<body>

<asp:Image id="imgLogo" runat="server"
 ImageUrl="<%# GetImageUrl()%>"></asp:Image><br>

</body></html>
```

Wenn der für die Datenbindung verwendete Ausdruck Anführungszeichen enthält, meldet ASP.NET bei der Ausführung der ASPX*-Seite den Fehler »Servertag wurde falsch formatiert« . Dieser Fehler wird z. B. generiert, wenn ein ASP-Button seinen Text aus einer Auflistung auslesen soll:* `<asp:Button Text="<%# ht.Item("Button1") %>" runat="server"/>`. *Die ASP.NET-Engine kommt scheinbar nicht mit den mehrfachen Anführungszeichen zurecht. Ersetzen Sie die zum Tag gehörenden Anführungszeichen dann durch einfache Apostrophe:* `<asp:Button Text='<%# ht.Item("Button1") %>' runat="server"/>`.

Die Einfachwert-Datenbindung wird hauptsächlich bei Steuerelementen wie dem `Repeater`- und dem `DataList`-Steuerelement eingesetzt, die lediglich eine von Ihnen bestimmte HTML-Vorlage für jede Zeile der Datenquelle wiederholen. Ab Seite 409 finden Sie dazu nähere Informationen.

14.1.2 Mehrzeilen-Datenbindung

Acht der ASP-Steuerelemente können nicht nur einen einzelnen Wert, sondern gleich eine Liste von Daten anzeigen. Dazu gehören die folgenden Steuerelemente:

- das `HtmlSelect`-Steuerelement,
- die ASP-`Listbox`,
- das ASP-`DropDownList`-Steuerelement,
- das ASP-`CheckBoxList`-Steuerelement,
- das ASP-`RadioButtonList`-Steuerelement,
- das ASP-`Repeater`-Steuerelement,
- das ASP-`DataList`-Steuerelement und
- das ASP-`DataGrid`-Steuerelement.

Lediglich das `Repeater`-, das `DataList`- und das `DataGrid`-Steuerelement sind in der Lage, eine mehrspaltige Liste anzuzeigen. Alle anderen arbeiten nur mit einer einspaltigen Liste (z. B. einem Array). Sofern Sie eine Liste mit mehreren Spalten (z. B. ein `DataView`-Objekt

mit den Daten einer Datenbankabfrage) an diese Steuerelemente zuweisen, müssen Sie in der Eigenschaft `DataTextField` festlegen, welche Spalte angezeigt werden soll.

Die gemeinsamen Eigenschaften

Die Steuerelemente für Listen-Datenbindung besitzen einige gemeinsame Eigenschaften, die die Datenbindung steuern.

Eigenschaft	Beschreibung
`DataTextField`	beschreibt bei mehrspaltigen Datenquellen, welche Spalte angezeigt werden soll
`DataValueField`	bestimmt, welche Spalte als Werte des einzelnen Eintrags verwendet werden soll. Dieser Wert wird in das `value`-Attribut des HTML-Elements eingetragen. Hier können Sie z. B. eine eindeutige Artikelnummer speichern, wenn Sie in der Liste den Artikelnamen anzeigen. Dies erleichtert den späteren Zugriff auf die Daten.
`DataText-FormatString`	Hier können Sie einen Formatier-String angeben, der dann zur Formatierung verwendet wird, wenn die Daten angezeigt werden. Gültig sind die unter .NET üblichen Formatierstrings. "{o:C}" steht z. B. für einen Currency-Wert, "{o:dd.mm.yy}" für ein Datum in der kurzen Form.
`DataMember`	Datenquellen können aus mehreren Sätzen (Rowsets) bestehen, die unterschiedliche Daten speichern. Das kommt eher selten vor, z. B. beim Aufruf von Stored Procedures. Ein Satz Daten enthält dann z. B. die Artikel einer Bestelldatenbank, ein weitere enthält die Artikel-Kategorien. Besteht die Datenquelle aus mehreren Sätzen, geben Sie in `DataMember` den zu verwendenden Satz an.

Tabelle 14.1: Die gemeinsamen Eigenschaften der Steuerelemente für Listen-Datenbindung

Methode	Beschreibung
`DataBind`	fordert das Steuerelement auf, sich mit den Daten der Datenquelle zu füllen
`FindControl`	Über diese Methode können Sie eine Referenz auf ein Element ermitteln, das auf dem Steuerelement (automatisch) angelegt wurde. Damit können Sie zum Beispiel den Wert einer bestimmten Tabellenzelle auslesen.

Tabelle 14.2: Die wichtigen gemeinsamen Methoden der Steuerelemente für Listen-Datenbindung

Ereignis	Beschreibung
`DataBinding`	wird für jede Zeile der Datenquelle aufgerufen, bevor die Zeile vom Steuerelement ausgewertet wird. Hier können Sie den Inhalt der Zeile bearbeiten und damit die Darstellung der Daten manipulieren.
`Selected-IndexChanged`	wird aufgerufen, wenn der aktuell ausgewählte Eintrag geändert wurde

Tabelle 14.3: Die wichtigen gemeinsamen Ereignisse der Steuerelemente für Listen-Datenbindung

Datenquellen anbinden

Eine Datenquelle muss lediglich ein Objekt sein, das die Schnittstelle `ICollection`, `IEnumerable` oder `IListSource` implementiert. Die Auflistungs-Klassen des `Collection`-Namensraums (`ArrayList`, `HashTable`, `SortedList` etc.) implementieren mindestens eine dieser Schnittstellen, aber auch die Datenzugriffs-Klassen des `System.Data`-Namensraums (`DataSet`, `DataReader` und `DataView`). Für die Datenbindung ist es im Prinzip unerheblich, ob eine Auflistung, eine ADO-Datenquelle oder ein anderes Objekt an das Steuerelement gebunden ist. Hauptsache ist, das Objekt implementiert eine der genannten Schnittstellen.

Das Anbinden der Datenquelle ist einfach. Setzen Sie eine Referenz auf das Objekt, das die Daten speichert, auf die Eigenschaft `DataSource` und rufen Sie danach die `DataBind`-Methode auf. Das folgende

Beispiel erzeugt ein `ArrayList`-Objekt, füllt dieses und bindet es dann an eine ASP-Listbox an:

```
<script language="VB" runat="Server">
 Private Sub Page_Load()
   ' ArrayList erzeugen
   Dim arr As ArrayList = New ArrayList(10)
   arr.Add("Nitty-Gritty Visual Basic 6")
   arr.Add("Nitty-Gritty Programmierung")
   arr.Add("Nitty-Gritty Internetprogrammierung")
   arr.Add("Goto Internetprogrammierung")
   arr.Add("Nitty-Gritty C#")
   arr.Add("Nitty-Gritty Visual Basic.NET")
   arr.Add("Programmieren lernen")

   ' ArrayList an die Listbox anbinden
   lstBooks.DataSource = arr
   ' Listbox auffordern, die Daten einzulesen
   lstBooks.DataBind()

 End Sub
</script>
```

Einige Auflistungen der BCL bestehen aus einem Schlüssel-Wert-Paar. Ein Schlüssel identifiziert jeweils die einzelnen Werte. Die `HashTable`-Klasse ist ein Beispiel dafür. Wenn Sie diese Auflistungen anbinden wollen, müssen Sie sich entscheiden, ob Sie die Werte oder die Schlüssel anzeigen wollen. Binden Sie dazu dann entweder die `Values`- oder die `Keys`-Eigenschaft an:

```
' Hashtable erzeugen
Dim ht as HashTable = New HashTable()
ht.Add(1, "Zaphod")
ht.Add(2, "Trillian")
ht.Add(3, "Ford")
ht.Add(4, "Arthur")
' Bei einer Hashtable muss entweder die Eigenschaft
' Values oder Keys gebunden werden
lstPersons.DataSource = ht.Values
lstPersons.DataBind()
```

Wenn Sie einmal aus Versehen die `Values`*- oder die* `Keys`*-Eigenschaft nicht angeben, reagiert Ihr Programm mit einer nicht besonders aussagekräftigen Ausnahme »xyz mit der ID 'abc' konnte die Spalten von der ausgewählten Datenquelle nicht automatisch generieren«.*

14.1.3 Stilvorlagen

Die speziellen datengebundenen Steuerelemente wie das `DataGrid`- und das `DataList`-Steuerelement erlauben so genannte Vorlagen (Templates), die das Aussehen bestimmter Teile des Steuerelements bestimmen. Diese Vorlagen können Sie mit Visual Studio recht einfach erzeugen. Aktivieren Sie dazu das Steuerelement und klicken Sie dann auf den Link AUTOM. FORMATIERUNG im Eigenschaftenfenster. Im erscheinenden Dialog können Sie aus einer Vielzahl vorgefertigter Formatvorlagen eine passende auswählen (Abbildung 14.1).

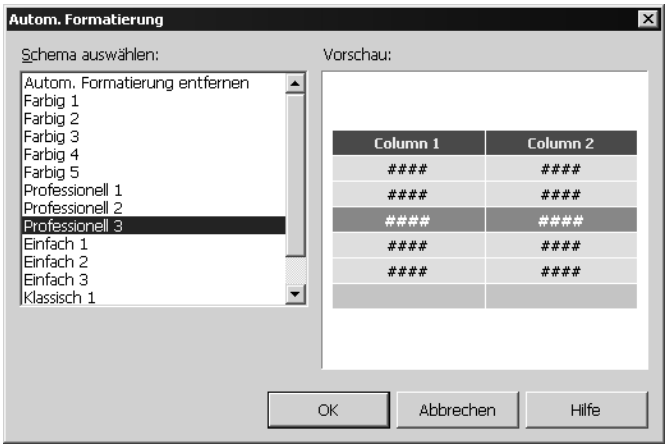


Bild 14.1: (Halb-)Automatische Erzeugung einer Vorlage für ein DataGrid

Alternativ und zur nachträglichen Feineinstellung können Sie die Vorlage (neben den anderen Eigenschaften des Steuerelements) im Eigenschaftengenerator bearbeiten, den Sie über den entsprechenden Link im Eigenschaftenfenster öffnen.

Visual Studio erzeugt entsprechende Tags für die Vorlage. Ein `DataGrid`-Steuerelement wird dann beispielsweise im Quelltext folgendermaßen dargestellt:

```
<asp:datagrid id="DataGrid1" runat="server"
 CellSpacing="1" CellPadding="3" GridLines="None"
 BorderWidth="2px" BorderColor="White" BorderStyle="Ridge"
 BackColor="White">
 <FooterStyle ForeColor="Black"
  BackColor="#C6C3C6"></FooterStyle>
 <HeaderStyle Font-Bold="True" ForeColor="#E7E7FF"
  BackColor="#4A3C8C"></HeaderStyle>
 <PagerStyle HorizontalAlign="Right" ForeColor="Black"
  BackColor="#C6C3C6"></PagerStyle>
 <SelectedItemStyle Font-Bold="True" ForeColor="White"
  BackColor="#9471DE"></SelectedItemStyle>
 <ItemStyle ForeColor="Black"
  BackColor="#DEDFDE"></ItemStyle>
</asp:datagrid>
```

14.1.4 Das DataGrid-Steuerelement

Das `DataGrid`-Steuerelement zeigt eine mehrspaltige Datenquelle in Tabellenform. Es bietet Ihnen viele Möglichkeiten, mehrspaltige Daten auf eine einfache Weise anzuzeigen. Dieses Steuerelement wird hauptsächlich mit ADO.NET-Datenquellen verwendet. ADO.NET wird zwar erst ab Seite 423 behandelt, die Technik ist aber sehr ähnlich zu dem in Kapitel 10 behandelten ADO.

Das folgende Beispiel erzeugt ein `DataReader`-Objekt (das verwendet wird, um genau einmal durch eine Datenquelle gehen zu können) mit den Daten der Artikeltabelle der Nordwind-Datenbank:

```
' Öffnen der Verbindung zur Nordwind-Datenbank
' mit Hilfe einer udl-Datei, die die Verbindungs-
' Informationen speichert
Dim con As Data.OleDb.OleDbConnection
con = New Data.OleDb.OleDbConnection("File Name=" & _
   MapPath(".") & "\..\..\Nordwind.udl")
con.Open()
```

```
' DataReader erzeugen
Dim cmd As Data.OleDb.OleDbCommand
Dim strSQL As String
strSQL = "SELECT [Artikel-Nr], Artikelname, Einzelpreis From Artikel
ORDER BY Artikelname"
cmd = New Data.OleDb.OleDbCommand(strSQL, con)
Dim dr As Data.OleDb.OleDbDataReader
dr = cmd.ExecuteReader()

' DataReader an das DataGrid anbinden
dgdProducts.DataSource = dr
dgdProducts.DataBind()
```

Das Ergebnis zeigt Abbildung 14.2. Die Darstellung des Grid ist zwar noch nicht allzu schön, aber daran können wir noch einiges ändern.

Die Artikel aus der Nordwind-Datenbank:

Artikel-Nr	Artikelname	Einzelpreis
17	Alice Mutton	39
3	Aniseed Syrup	10
40	Boston Crab Meat	18,4
60	Camembert Pierrot	34
18	Carnarvon Tigers	62,5
1	Chai	18
2	Chang	19
39	Chartreuse verte	18
4	Chef Anton's Cajun Seasoning	22

Bild 14.2: Ein DataGrid-Steuerelement, das mit den Daten einer Abfrage gefüllt wurde

Das folgende Beispiel ändert die Formatierung des DataGrid-Steuerelements leicht ab, sodass die Anzeige etwas schöner wird:

```
<asp:DataGrid
 id="dgdProducts"
 BackColor="darkgray"
 BackImageUrl="background.jpg"
 BorderStyle="solid"
 BorderColor="black"
```

```
 BorderWidth="3px"
 GridLines="None"
 CellPadding="2"
 CellSpacing="2"
 Font-Name="Verdana"
 Font-Size="10pt"
 Font-Bold="true"
 runat="server">
</asp:DataGrid>
```

Artikel-Nr	Artikelname	Einzelpreis
17	Alice Mutton	39
3	Aniseed Syrup	10
40	Boston Crab Meat	18,4
60	Camembert Pierrot	34
18	Carnarvon Tigers	62,5
1	Chai	18
2	Chang	19
39	Chartreuse verte	18
4	Chef Anton's Cajun Seasoning	22
5	Chef Anton's Gumbo Mix	21,35

Bild 14.3: Ein optisch ansprechenderes DataGrid

Festlegen, welche Spalten angezeigt werden

Normalerweise zeigt das `DataGrid` immer alle Spalten der Datenquelle in der Reihenfolge an, in der diese in der Datenquelle definiert sind. Sie können die automatische Erzeugung der Spalten jedoch auch ausschalten und die anzuzeigenden Spalten über `BoundColumn`-Elemente selbst definieren. Dabei können Sie die verschiedensten Attribute für die Spalte setzen. Im folgenden Beispiel wird die Artikelnummer ausgeblendet, der Preis wie eine Währung formatiert und rechtsbündig ausgegeben:

```
<asp:DataGrid
 id="dgdProducts"
 BackColor="darkgray"
 BackImageUrl="background.jpg"
 BorderStyle="solid"
 BorderColor="black"
```

```
 BorderWidth="3px"
 GridLines="None"
 CellPadding="2"
 CellSpacing="2"
 Font-Name="Verdana"
 Font-Size="10pt"
 Font-Bold="true"
 AutoGenerateColumns="false"
 runat="server">
 <columns>
   <asp:BoundColumn HeaderText="Artikel"
    DataField="Artikelname"/>
   <asp:BoundColumn HeaderText="Preis"
    DataField="Einzelpreis"
    DataFormatString="{0:#.00 €}"
    HeaderStyle-HorizontalAlign="Right"
    ItemStyle-HorizontalAlign="Right"/>
 </columns>
</asp:DataGrid>
```

Artikel	Preis
Alice Mutton	39,00 €
Aniseed Syrup	10,00 €
Boston Crab Meat	18,40 €
Camembert Pierrot	34,00 €
Carnarvon Tigers	62,50 €
Chai	18,00 €
Chang	19,00 €
Chartreuse verte	18,00 €
Chef Anton's Cajun Seasoning	22,00 €
Chef Anton's Gumbo Mix	21,35 €

Bild 14.4: Das DataGrid mit selbst definierten Spalten

Auswerten von Ereignissen am Beispiel des benutzerdefinierten Sortierens

Wenn Sie die Eigenschaft `AllowSorting` auf `True` einstellen, erzeugt das DataGrid automatisch Hyperlinks an den Überschriften der Spalten. Wenn der Anwender auf einen dieser Hyperlinks klickt, wird (über ein integriertes JavaScript-Programm) das Ereignis `SortCommand`

aufgerufen. Sie können dieses Ereignis direkt bei der Deklaration des `DataGrid`-Steuerelements über die `OnSortCommand`-Eigenschaft mit einer Ereignisprozedur verbinden:

```
<form id="frmDemo" runat="server">
Die Artikel aus der Nordwind-Datenbank:<br>
<asp:DataGrid
 id="dgdProducts"
 AllowSorting="true"
 OnSortCommand="SortProducts"
 BorderStyle="solid"
 BorderColor="black"
 BorderWidth="3px"
 GridLines="None"
 CellPadding="2"
 CellSpacing="2"
 runat="server">
<HeaderStyle BackColor="#E0E0E0"></HeaderStyle>
</asp:DataGrid>
</form>
```

Der Programmcode sieht nun so aus, dass das `SortCommand`-Ereignis eine separate Prozedur zum Anbinden des Steuerelements aufruft. Die ausgewählte Sortierung, die dieses Ereignis aus der Eigenschaft `SortExpression` des übergebenen Source-Arguments ausliest, wird dabei an die Anbinde-Prozedur weitergegeben. Damit das `DataGrid` beim ersten Laden der Seite auch gefüllt wird, wird diese Anbinde-Prozedur auch im `Load`-Ereignis der Seite aufgerufen:

```
<script language="VB" runat="Server">

 ' Load-Ereignisprozedur der Seite
 Private Sub Page_Load()
   If IsPostBack = False Then
      DataBind("Artikelname")
   End If
 End Sub

 ' Ereignisprozedur für das SortCommand-Ereignis
 Private Sub SortProducts( _
```

```
    ByVal source As Object, _
    ByVal e As WebControls.DataGridSortCommandEventArgs)

    ' Daten mit der neuen Sortierung neu anbinden
    DataBind(e.SortExpression.ToString())
End Sub

' Private Prozedur zum Anbinden der Daten
Private Sub DataBind(ByVal strSort As String)
 ' Öffnen der Verbindung zur Nordwind-Datenbank
 ' mit Hilfe einer udl-Datei, die die Verbindungs-
 ' Informationen speichert
 Dim con As Data.OleDb.OleDbConnection
 con = New Data.OleDb.OleDbConnection("File Name=" & _
    MapPath(".") & "\..\..\Nordwind.udl")
 con.Open()

 ' DataReader erzeugen
 Dim cmd As Data.OleDb.OleDbCommand
 Dim strSQL As String
 strSQL = "SELECT [Artikel-Nr], Artikelname, " & _
    "Einzelpreis From Artikel "
 If strSort > "" Then
    ' Wenn ein Sortierbegriff übergeben wurde,
    ' diesen an den SQL-String anhängen
    strSQL = strSQL & " ORDER BY [" & strSort & "]"
    ' Die eckigen Klammern werden für Access benötigt,
    ' wenn Felder wie "Artikel-Nr" Sonderzeichen im
    ' Namen tragen.
 End If

 cmd = New Data.OleDb.OleDbCommand(strSQL, con)
 Dim dr As Data.OleDb.OleDbDataReader
 dr = cmd.ExecuteReader()

 ' DataReader an das DataGrid anbinden
 dgdProducts.DataSource = dr
 dgdProducts.DataBind()
```

```
End Sub
</script>
```

Fertig ist das Sortieren.

Das DataGrid ist noch wesentlich mächtiger, als hier dargestellt. Es unterstützt über verschiedene Ereignisse das Editieren der Daten, ermöglicht die Manipulation der Daten vor der Anzeige derselben und erlaubt die Darstellung der Liste auf mehreren Seiten, die der Benutzer komfortabel auswählen kann. Hier ist nun aber leider kein Platz mehr für weitere Erläuterungen. In den Buchbeispielen zu diesem Kapitel finden Sie einige Beispiele, die alle Möglichkeiten des `DataGrid` *beleuchten.*

14.1.5 Das Repeater-Steuerelement

Als ich im Internet das erste Beispiel für das Repeater-Steuerelement gesehen hatte, dachte ich: »Mein Gott, was ist das für ein kompliziertes Steuerelement«. Aber es war gar nicht das Steuerelement, das so kompliziert war, sondern das Programm.

Funktionsweise

Das `Repeater`-Steuerelement ist eigentlich ganz einfach. Es wiederholt die HTML- und ASP-Elemente, die der Entwickler innerhalb einer Vorlage (einem »Template«) platziert, für jeden Eintrag der verbundenen Liste. Im Ergebnis erscheint dann eine Liste, die so formatiert ist, wie dies in der Vorlage beschrieben ist. Die Liste kann so definiert werden, dass diese mit drei Bereichen ausgeben wird: dem Kopf, dem Detailbereich und dem Fuß. Das Layout eines Bereichs wird einfach über eine separate Vorlage bestimmt. Die einzig notwendige Vorlage ist die für den Detailbereich.

Das `Repeater`-Steuerelement unterstützt neben dem bloßen Anzeigen der Daten auch das Editieren. Dazu werden dann weitere Vorlagen eingesetzt: eine für das Layout einer selektierten Zeile und eine für das Layout einer Zeile, die zur Bearbeitung freigegeben ist.

Ein Beispiel soll das Ganze etwas verdeutlichen. Es erzeugt im `Load`-Ereignis der Seite ein `DataReader`-Objekt mit den Daten der Artikel-Ta-

belle der Nordwind-Datenbank und bindet dieses an ein einfaches `Repeater`-Steuerelement. Dieses Steuerelement beinhaltet die einzig notwendige Vorlage `ItemTemplate`, die für jede Zeile der Auflistung ausgeführt wird:

```
<%@Page Language="VB"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">

<title>Repeater-Steuerelement</title>

<script language="VB" runat="Server">
 Private Sub Page_Load()
  ' Öffnen der Verbindung zur Nordwind-Datenbank
  ' Aus Vereinfachungsgründen nicht dargestellt
  ...

  ' DataReader an das Repeater-Steuerelement anbinden
  Repeater.DataSource = dr
  Repeater.DataBind()

 End Sub
</script>

</head>

<body>

<h1>Repeater-Steuerelement</h1>

<form id="Form1" method="post" runat="server">
 <asp:Repeater id="Repeater" runat="server">
    <ItemTemplate>
       <%# Container.DataItem("Artikel-Nr") %>
       <%# Container.DataItem("Artikelname") %><br>
    </ItemTemplate>
 </asp:Repeater>
</form>
```

```
</body>
</html>
```

Die Ausgabe dieses Programms ist (noch) recht einfach gestaltet (Abbildung 14.5).

Bild 14.5: Einfache Datenbindung mit dem Repeater-Steuerelement

Innerhalb der Vorlagen können Sie Text, HTML-Elemente, ASP-Elemente und CSS-Stile definieren. Zum Zugriff auf die aufgelisteten Daten verwenden Sie die Datenbindungssyntax für einfache Werte. Innerhalb der Vorlage arbeiten Sie ja nicht mehr mit einer Liste, sondern nur noch mit einer einzelnen Zeile, aus der Sie eine oder mehrere Spalten auslesen können.

Über die `DataItem`-Eigenschaft des `Container`-Objekts (`Container` verweist auf ein Objekt der Klasse `RepeaterItem`, das die einzelne Datenzeile verwaltet) erhalten Sie Zugriff auf die Datenzeile. Wenn die Datenquelle ein ADO.NET-Objekt ist, greifen Sie – wie im Beispiel – über den Schlüssel der in `DataItem` gelieferten Feld-Auflistung auf die Daten zu.

Handelt es sich bei der Datenquelle um eine einfache Auflistung mit Schlüssel (`HashTable`, `SortedList` etc), binden Sie diese direkt an (nicht über die `Values`-Eigenschaft):

```
Dim ht As HashTable = New HashTable
ht.Add(1, "Zaphod")
...
ht.Add(4, "Arthur")
Repeater.DataSource = ht
```

Dann können Sie über die Eigenschaften `Value` und `Key` auf die gespeicherten Werte zugreifen:

```
<asp:Repeater id="Repeater" runat="server">
   <ItemTemplate>
      <%# Container.DataItem.Key %>
      <%# Container.DataItem.Value %>
   </ItemTemplate>
</asp:Repeater>
```

Handelt es sich nur um eine einfache Auflistung, greifen Sie direkt über `DataItem` zu.

Die Vorlagen

Beim `Repeater`-Steuerelement können Sie fünf Vorlagen zur Gestaltung der Ausgabe benutzen:

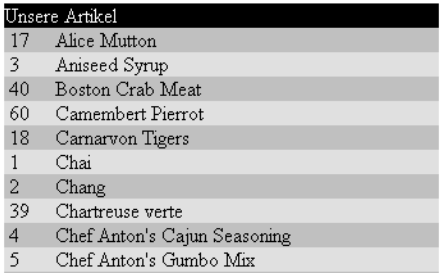
- `HeaderTemplate`: Diese Vorlage wird nur für den Kopf der Daten ausgeführt
- `ItemTemplate`: `ItemTemplate` wird für jede Datenzeile ausgeführt, wenn `AlternatingItemTemplate` nicht definiert ist, ansonsten für jede zweite Datenzeile
- `AlternatingItemTemplate`: Wenn Sie diese Vorlage definieren, führt das `Repeater`-Steuerelement für die einzelnen Zeilen im Wechsel die `ItemTemplate`- und die `AlternatingItemTemplate`-Vorlage aus
- `SeparatorTemplate`: Diese Vorlage wird zwischen den Datenzeilen ausgeführt. Hier können Sie z. B. einen »Trennbalken« zwischen den einzelnen Zeilen einrichten
- `FooterTemplate`: Vorlage für die Fußzeile

Das folgende Beispiel verdeutlicht den Umgang mit Vorlagen. Es bewirkt, dass eine Kopfzeile ausgegeben wird und dass die Datenzeilen abwechselnd mit grauem und weißem Hintergrund gezeichnet werden. Für die Positionierung und Formatierung verwende ich einfach ASP-Label:

```
<asp:Repeater id="Repeater" runat="server">
  <HeaderTemplate>
    <asp:Label Width="340" BackColor="Black"
    Style="position:absolute;left:10"
    ForeColor="White" Runat="Server">
     Unsere Artikel
   </asp:Label>
   <br>
  </HeaderTemplate>

  <ItemTemplate>
    <asp:Label Style="position:absolute;left:10"
      Width="40" BackColor="Silver" runat="Server">
      <%# " " & Container.DataItem("Artikel-Nr") %>
    </asp:Label>
    <asp:Label Style="position:absolute;left:50"
      Width="300" BackColor="Silver" Runat="Server">
      <%# Container.DataItem("Artikelname") %>
    </asp:Label>
    <br>
  </ItemTemplate>

  <AlternatingItemTemplate>
    <asp:Label Style="position:absolute;left:10"
      Width="40" BackColor="#E0E0E0" runat="Server">
      <%# " " & Container.DataItem("Artikel-Nr") %>
    </asp:Label>
    <asp:Label Style="position:absolute;left:50"
      Width="300" BackColor="#E0E0E0" Runat="Server">
      <%# Container.DataItem("Artikelname") %>
    </asp:Label>
    <br>
  </AlternatingItemTemplate>
</asp:Repeater>
```



Unsere Artikel	
17	Alice Mutton
3	Aniseed Syrup
40	Boston Crab Meat
60	Camembert Pierrot
18	Carnarvon Tigers
1	Chai
2	Chang
39	Chartreuse verte
4	Chef Anton's Cajun Seasoning
5	Chef Anton's Gumbo Mix

Bild 14.6: Ein mit Stilen und Vorlagen verfeinertes Repeater-Steuerelement

Statt ASP-Steuerelemente für die Anzeige der Daten zu verwenden, können Sie diese (natürlich) auch direkt ausgeben und über CSS formatieren:

```
<ItemTemplate>
 <span style="position:absolute;left:10;
  width:40;background-color:Silver">
  <%# " " & Container.DataItem("Artikel-Nr") %>
 </span>
 <span style="position:absolute;left:50;width:300;
  background-color:Silver">
  <%# Container.DataItem("Artikelname") %>
 </span>
 <br>
 </ItemTemplate>
```

Dann müssen Sie sich aber ein wenig mehr mit CSS und der Browserkompatibilität herumschlagen.

Das ItemCommand-Ereignis

Das wichtigste Ereignis des `Repeater`-Steuerelements ist `ItemCommand`. Dieses Ereignis wird aufgerufen, wenn ein Steuerelement, das in einer Vorlage des `Repeater`-Steuerelements angelegt ist, einen Submit des Formulars auslöst. Sie können damit z. B. auf die Betätigung eines `Button` oder eines `LinkButton` innerhalb des `Repeater`-Steuerelements reagieren. Die Ereignisprozedur besitzt die folgende Signatur:

```
Private Sub Name(ByVal source As System.Object, _
   ByVal e As _
   System.Web.UI.WebControls.RepeaterCommandEventArgs)
```

Das Argument *source* zeigt, wie bei allen Ereignisprozeduren, auf das auslösende Objekt, das allerdings das `Repeater`-Steuerelement selbst ist. Das Argument *e* ist ein Objekt der Klasse `RepeaterCommandEventArgs`. Die Eigenschaften CommandName und CommandArgument dieses Objekts geben Ihnen Informationen über die auszuführende Aktion. Die Werte dieser Eigenschaften definieren Sie selbst in den gleichnamigen Eigenschaften der auslösenden Steuerelemente. Die Eigenschaft `CommandSource` ist eine Referenz auf das auslösende Steuerelement.

Das folgende Beispiel (das auf den vorherigen Beispielen aufbaut) verdeutlicht den Umgang mit diesem Ereignis. Ich zeige zunächst das Ergebnis der Programmierung. Wenn die ASPX-Seite angefordert wird, soll der Artikelname über einen `LinkButton` dargestellt werden.

Bild 14.7: Ein Repeater-Steuerelement mit LinkButtons

Klickt der Anwender auf einen Link, soll über der Liste ein `Panel` sichtbar werden, in dem der Anwender Detaildaten sieht und den Artikel bestellen kann.

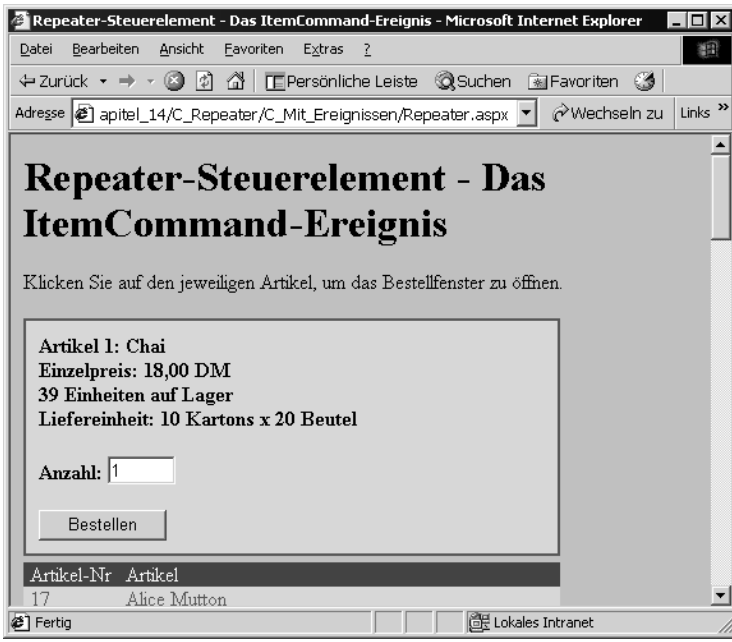


Bild 14.8: Ein Panel wird sichtbar, nachdem der Anwender auf einen LinkButton geklickt hat

Zur Lösung dieses Problems habe ich zunächst oben auf der Seite einen Stil für die Hyperlinks erzeugt, weil diese in bestimmten Farben ausgegeben werden sollen (das LinkButton-Steuerelement besitzt keine entsprechenden Eigenschaften):

```
<%@Page Language="VB"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
```

```
<title>Repeater-Steuerelement - Das ItemCommand-Ereignis</title>

<!-- Stil für die Links in den Datenzeilen des Repeater -->
<style>
 a.Item:link {color:#3F3F65}
 a.Item:visited {color:#3F3F65}
 a.Item:active {color:black}
 a.AlternatingItem:link {color:#D6D6E4}
 a.AlternatingItem:visited {color:#D6D6E4}
 a.AlternatingItem:active {color:black}
</style>
```

Im `Load`-Ereignis der Seite wird wieder die Datenbankverbindung geöffnet und die Artikeltabelle ausgelesen. Eine kleine Änderung ist allerdings notwendig, da die Verbindung auch im `ItemCommand`-Ereignis benötigt wird: Die Deklaration der Connection-Variable wird oberhalb der Prozedur vorgenommen:

```
<script language="VB" runat="Server">

 Private con As Data.OleDb.OleDbConnection

 Private Sub Page_Load()
  ' Öffnen der Verbindung zur Nordwind-Datenbank
  ...
 End Sub
```

Damit die Verbindung sauber wieder geschlossen wird, erledige ich das im `Unload`-Ereignis der Seite:

```
 Private Sub Page_Unload
  con.Close()
 End Sub
```

Bevor ich die wichtige Ereignisprozedur für das `ItemCommand`-Ereignis erläutere, zeige ich erst einmal, wie der Inhalt der Seite deklariert ist. Im oberen Bereich der Seite ist das Panel deklariert. Dieses ist aber erst einmal unsichtbar geschaltet:

```
<body bgColor="Silver">
```

```
<h1>Repeater-Steuerelement - Das ItemCommand-Ereignis</h1>
Klicken Sie auf den jeweiligen Artikel, um das Bestellfenster zu öff-
nen.
<form id="frm" runat="server">

 <!-- Panel für die Artikel-Details -->
 <asp:Panel id="pnlArtikelDetails"
  Width="420" Height="120px" BackColor="#D6D6E4"
  ForeColor="Black" BorderStyle="Solid"
  BorderColor="#535386" BorderWidth="2px"
  Font-Bold="true" Visible="False"
  Style="padding:10;margin-bottom:5" runat="Server">
   <asp:Label id="lblArtikel" runat="server"/><br>
   <asp:Label id="lblArtikelPreis" runat="server"/><br>
   <asp:Label id="lblArtikelMenge" runat="server"/><br>
   <asp:Label id="lblArtikelLiefereinheit"
    runat="server"/>
   <p>Anzahl: <asp:Textbox id="txtAnzahl" Text="1"
    Size="3" runat="server"/></p>
   <p><asp:Button Text="Bestellen" width="100px"
    OnClick="HandleBestellen" runat="server"/></p>
 </asp:Panel>
```

Danach folgt das Repeater-Steuerelement. Das `ItemCommand`-Ereignis wird über die `OnItemCommand`-Eigenschaft mit der Ereignisprozedur verbunden. Für den Artikelnamen wird ein LinkButton verwendet. Damit der betätigte Button im `ItemCommand`-Ereignis identifiziert werden kann, ist für jeden Link-Button die Eigenschaft `CommandName` und `CommandArgument` definiert. Auf CommandName könnte hier zwar verzichtet werden, weil das Programm nur einen »Befehl« zulässt. Ich setze diese Eigenschaft aber trotzdem ein, damit ich bei späteren Erweiterungen des Programms keine Probleme habe. `CommandArgument` wird auf die jeweilige Artikel-Nummer gesetzt, um den gewählten Artikel identifizieren zu können:

```
 <asp:Repeater id="Repeater"
  OnItemCommand="HandleItemCommand"
  runat="server">
  <HeaderTemplate>
```

```
  <asp:Label Style="position:absolute;left:10;
    padding-left:5" Width="80" ForeColor="White"
    BackColor="#3F3F65" Runat="Server"
    Text="Artikel-Nr"/>
  <asp:Label Style="position:absolute;left:90"
   Width="340" ForeColor="White"
   BackColor="#3F3F65" Runat="Server"
   Text="Artikel"/>
 <br>
</HeaderTemplate>

<ItemTemplate>
  <asp:Label Width="80" ForeColor="#535386"
    BackColor="#D6D6E4"
      Style="position:absolute;left:10;padding-left:5"
   Runat="Server">
    <%# Container.DataItem("Artikel-Nr") %>
  </asp:Label>
  <asp:LinkButton class="AlternatingItem"
   Style="position:absolute;left:90" Width="340"
    ForeColor="#535386" BackColor="#D6D6E4"
   Text='<%# Container.DataItem("Artikelname") %>'
    CommandName="Show"
    CommandArgument=
      '<%# Container.DataItem("Artikel-Nr") %>'
    runat="server"/>
  <br>
</ItemTemplate>

<AlternatingItemTemplate>
  <asp:Label Style="position:absolute;left:10;
    padding-left:5" Width="80" ForeColor="#D6D6E4"
    BackColor="#535386" runat="Server">
    <%# Container.DataItem("Artikel-Nr") %>
  </asp:Label>
  <asp:LinkButton class="AlternatingItem"
   Style="position:absolute;left:90" Width="340"
    ForeColor="#D6D6E4" BackColor="#535386"
```

```
      Text='<%# Container.DataItem("Artikelname") %>'
      CommandName="Show"
      CommandArgument=
        '<%# Container.DataItem("Artikel-Nr") %>'
      runat="server"/>
    <br>
  </AlternatingItemTemplate>
</asp:Repeater>
```

Die `ItemCommand`-Ereignisprozedur wertet den Befehl und das Befehlsargument aus, liest die Daten des gewählten Artikels aus, schreibt diese in die Label auf dem Panel und schaltet das Panel sichtbar:

```
Private Sub HandleItemCommand( _
   ByVal source As System.Object, _
   ByVal e As WebControls.RepeaterCommandEventArgs)
 ' Ereignisprozedur für das Ereignis
 ' ItemCommand des Repeater-Steuerelements

 If e.CommandName = "Show" Then
    ' DataReader erzeugen
    Dim cmd As Data.OleDb.OleDbCommand
    Dim strSQL As String
    strSQL = "SELECT * FROM Artikel " & _
       "WHERE [Artikel-Nr] = " & e.CommandArgument
    cmd = New Data.OleDb.OleDbCommand(strSQL, con)
    Dim dr As Data.OleDb.OleDbDataReader
    dr = cmd.ExecuteReader()
    ' Daten einlesen
    dr.Read()
    ' Daten auslesen und im Detail-Panel ausgeben
    lblArtikel.Text = "Artikel " & e.CommandArgument & _
       ": " & dr.Item("Artikelname").ToString()
   lblArtikelPreis.Text = "Einzelpreis: " & _
      String.Format("{0:C2}", dr.Item("Einzelpreis"))
   lblArtikelMenge.Text = dr.Item("Lagerbestand") & _
      " Einheiten auf Lager"
   lblArtikelLiefereinheit.Text = "Liefereinheit: " & _
      dr.Item("Liefereinheit")
```

```
    ' Panel sichtbar schalten
    pnlArtikelDetails.Visible = True
  End If
End Sub
```

Damit schließe ich das `Repeater`-Steuerelement ab. Sie können mit diesem Steuerelement mit recht wenig Aufwand gute Ergebnisse erzielen. Leider konnte ich nicht alle Möglichkeiten vorstellen.

Schauen Sie einmal in den Beispielen dieses Kapitels nach. Dort finden Sie `aspx`*-Seiten, die die Möglichkeiten dieses Steuerelements zeigen.*

14.1.6 Das DataList-Steuerelement

Das `DataList`-Steuerelement funktioniert ähnlich dem `Repeater`-Steuerelement. Es ist allerdings um einige Möglichkeiten erweitert:

- Die Liste kann mehrspaltig ausgegeben werden. Die Zeilen werden dann automatisch so umgerechnet, dass alle Spalten mit Zeilen gefüllt sind,
- die Liste kann in vertikaler oder horizontaler Richtung (ähnlich einem Kalender) verlaufen,
- es kann Stilvorlagen verwenden (Seite 402),
- es unterstützt den Entwickler dabei, Funktionen zum Editieren oder Löschen der Daten zu implementieren,
- die Liste kann so konfiguriert werden, dass das Selektieren einer Zeile möglich ist.

Zusätzlich zu den Vorlagen des `Repeater`-Steuerelements kann das `DataList`-Steuerelement die folgenden Vorlagen benutzen:

- `SelectedItemTemplate`: Vorlage für die selektierte Zeile,
- `EditItemTemplate`: Vorlage für Zeilen, die zur Bearbeitung freigegeben sind.

Der Unterschied zwischen dem `Repeater`- und dem `DataList`-Steuerelement ist, dass das `Repeater`-Steuerelement Ihnen mehr Freiheit bei der Ausgabe und der Bearbeitung der Daten lässt. Das `DataList`-Steu-

erelement ist dagegen einfacher anzuwenden, weil es viele Aufgaben bereits selbst erledigt.

Die Programmierung unterscheidet sich zunächst nicht vom `Repeater`-Steuerelement. Sie arbeiten wieder mit Vorlagen. Im Unterschied zum Repeater können Sie hier aber Stilvorlagen einsetzen (siehe Seite 402) und die Liste mehrspaltig formatieren:

```
<form name="frm" runat="server">
 <asp:DataList id="dl" RepeatColumns="3"
   CellSpacing="0" CellPadding="1" Runat="server">

   <HeaderStyle Font-Bold="True" ForeColor="White"
     BackColor="Gray"></HeaderStyle>
   <ItemStyle ForeColor="Black"
     BackColor="Silver"></ItemStyle>
   <AlternatingItemStyle ForeColor="Black"
     BackColor="#E0E0E0"></AlternatingItemStyle>

   <HeaderTemplate>
     Unsere Artikel
   </HeaderTemplate>

   <ItemTemplate>
    <%# Container.DataItem("Artikel-Nr") %>
    <%# Container.DataItem("Artikelname") %>
    <br>
   </ItemTemplate>

   <AlternatingItemTemplate>
    <%# Container.DataItem("Artikel-Nr") %>
    <%# Container.DataItem("Artikelname") %>
    <br>
   </AlternatingItemTemplate>
 </asp:DataList>
</form>
```

Die Eigenschaft `RepeatColumns` sorgt dafür, dass die Liste in mehreren Spalten angezeigt wird.

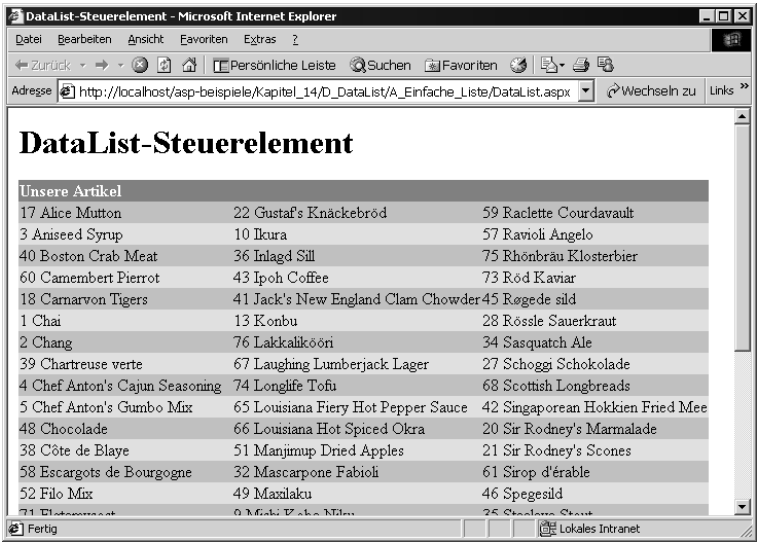


Bild 14.9: Eine dreispaltige Liste im DataList-Steuerelement

Das `DataList`*-Steuerelement kann noch einiges mehr als hier dargestellt. So unterstützt es zum Beispiel das Editieren und Löschen der dargestellten Daten. Leider ist hier nicht genug Platz für weitere Erläuterungen. In den Beispielen des Buchs finden Sie aber einige gut kommentierte* `aspx`*-Seiten, die die Möglichkeiten des DataList-Steuerelements demonstrieren.*

14.2 ADO.NET

Nachdem Sie das Prinzip der Datenbindung kennen gelernt und auch bereits mit einigen Datenbankzugriffen gearbeitet haben, zeige ich nun die Prinzipien des Datenzugriffs in .NET. Ich gehe dabei allerdings nur auf die für ASP.NET wichtigen Aspekte ein. Die Möglichkeiten, die ADO.NET bietet, sind wieder einmal so vielfältig, dass nur ein eigenes Buch alle Features beschreiben kann (ich glaube, gerade hatte ich eine Idee für ein neues Buch).

14.2.1 Das Prinzip des Datenzugriffs

Ähnlich ADO sieht ADO.NET alles das, was irgendwie Daten liefern kann, als Datenquelle an. Dazu gehören natürlich Datenbanken, aber auch z. B. E-Mail-Systeme, Verzeichnisdienste (wie LDAP), Office-Dokumente, Index-Dienste (die z. B. der Microsoft Index Server anbietet) etc.

Der Provider

Zum Zugriff auf die Daten benötigen Sie wie bei ADO einen Provider. Ein Hersteller einer Datenquelle kann diesen Provider zur Verfügung stellen. Anders als bei ADO wird der Provider aber in Form von mehreren Klassen direkt in das .NET-Framework integriert. ADO.NET liefert einen Provider für den SQL Server im Namensraum `System.Data.SqlClient`. Hier finden Sie alle Klassen, die Sie zur Arbeit mit dem SQL Server benötigen. Im Namensraum `System.Data.OleDb`[1] stellt ADO aber auch Klassen für den Zugriff auf Datenquellen zur Verfügung, für die auf dem System nur ein ADO-Provider vorhanden ist. Wahrscheinlich werden die Hersteller namhafter Datenbanksysteme wie Oracle aber bald auch echte ADO.NET-Provider für den direkten Zugriff auf ihre Datenquellen anbieten.

HINWEIS

Sie sollten wenn möglich immer den Provider für den direkten Zugriff verwenden. Der SQL Server-Provider nutzt z. B. direkt das API des SQL Servers zum Zugriff auf die Daten, ein Oracle-Provider wird das API von Oracle direkt nutzen. Gehen Sie stattdessen über den alten ADO-Provider, verwenden Sie eine weitere Zugriffsschicht (die den Zugriff nicht gerade beschleunigt). Außerdem ist ein ADO.NET-Provider wesentlich einfacher aufgebaut als ein ADO-Provider und wird deshalb performanter und ressourcenschonender arbeiten.

Die einzelnen Klassen der Provider müssen bestimmte Schnittstellen implementieren. Deswegen ist die Basisfunktionalität dieser Klassen immer gleich. Die Klasse, die zur Erzeugung einer Verbindung zur Da-

1. Der Name »OleDb« resultiert daraus, dass die ADO-Basistechnologie so genannt wird.

tenquelle verwendet wird, muss z. B. die Schnittstelle `IDbConnection` implementieren. Die Eigenschaften und Methoden, die in dieser Schnittstelle beschrieben sind, finden sich deshalb in allen Klassen zur Erzeugung einer Verbindung, sei es die Klasse `SqlConnection` zum Zugriff auf den SQL Server oder die Klasse `OleDbConnection` zum Zugriff auf ADO-Datenquellen. Die Implementierung dieser Eigenschaften und Methoden ist in den verschiedenen Klassen natürlich vollkommen unterschiedlich.

Einige der grundlegenden Klassen zum Zugriff auf die Daten sind allerdings direkt im Namensraum `System.Data` gespeichert. Dazu gehören im Wesentlichen die Klassen `DataView`, `DataSet` und `DataTable`.

Unterschiede zu ADO

ADO.NET besitzt viele Ähnlichkeiten mit ADO. Eine Verbindung zu einer Datenbank wird z. B. in beiden Varianten ähnlich aufgebaut. Aber ADO.NET ist auch in vielen Dingen unterschiedlich. Die folgende Auflistung hilft Ihnen dabei, die Unterschiede zu erfassen, wenn Sie bereits ein erfahrener ADO-Programmierer sind:

- ADO.NET besitzt keine Klasse, die mit der `Recordset`-Klasse identisch ist. Die ADO-Recordset-Klasse ist mit einer Vielzahl von Funktionen überladen. ADO.NET splittet diese Funktionalität auf in die Klassen `DataReader`, `DataTable` und `DataSet`,
- ADO.NET verwendet nur für die DataReader-Objekte so etwas Ähnliches wie einen clientseitigen Cursor. Das `DataSet`-Objekt arbeitet ohne Verbindung zur Datenquelle und benötigt deswegen auch keinen Cursor. Serverseitige Cursor sind unter ADO.NET gar nicht mehr möglich,
- Die Sperrmechanismen einer Datenbank werden nicht unterstützt. Das liegt daran, dass die Daten prinzipiell verbindungslos verwaltet werden,
- Wenn Sie gelesene Daten persistent in einer Datei abspeichern, geschieht dies nun immer in Form von XML.

14.2.2 Die fundamentalen Datenzugriffs-Klassen

Das .NET-Framework enthält eine Vielzahl von Klassen für den Datenzugriff. Viele dieser Klassen dienen allerdings lediglich als Datentyp

von Eigenschaften der fundamentalen Datenzugriffsklassen. Die Klasse `DataRow` repräsentiert z. B. eine Zeile, die von einem Objekt der Klasse `DataRowCollection` aufgelistet wird, das wiederum eine Eigenschaft eines `DataTable`-Objekts ist.

Die Verbindungs-Klassen

Über Objekte der Klassen `SqlConnection` und `OleDbConnection` bauen Sie eine Verbindung zur Datenquelle auf. Dazu übergeben Sie den Klassen bei der Erzeugung der Objekte oder später in der Eigenschaft `ConnectionString` einen Verbindungsstring. Über die `Open`-Methode öffnen Sie die Verbindung, die `Close`-Methode schließt die Verbindung wieder.

Die Befehls-Klassen

Um mit Daten arbeiten zu können, müssen Sie Befehle gegen die Datenbank ausführen. Zu diesem Zweck stellt Ihnen ADO.NET die Klassen `SqlCommand` und `OleDbCommand` zur Verfügung. Den Befehl (bei relationalen Datenquellen ist das eine SQL-Anweisung) übergeben Sie entweder direkt beim Erzeugen des Objekts oder später in der Eigenschaft `CommandText`. Die Methode `ExecuteReader` gibt ein Objekt der Klasse `SqlDataReader` bzw. `OleDbDataReader` zurück, wenn der Befehl eine Abfrage darstellt. Über dieses Objekt können Sie Daten lesen. Die Methode `ExecuteNonQuery` führt Befehle aus, die keine Abfragen sind.

Die DataReader-Klassen

Die `ExecuteReader`-Methode eines Befehlsobjekts gibt ein Objekt der Klasse `SqlDataReader` bzw. `OleDbDataReader` zurück. Mit Hilfe dieser Objekte können Sie einmal durch die ermittelten Daten gehen und diese auslesen. Das Objekt bleibt während des Lesens mit der Datenquelle verbunden, bis es geschlossen wird. Wenn Sie im Programm auf einen Datensatz zugreifen, liest das Objekt den Datensatz ein. Es benötigt deswegen sehr wenig Ressourcen. Die Verbindung kann allerdings so lange nicht durch andere Teile des Programms verwendet werden, wie das DataReader-Objekt die Daten liest.

Die wichtigsten Eigenschaften und Methoden dieser Klassen fasst Tabelle 14.4 zusammen:

Eigenschaft / Methode	Beschreibung
`Read()`	bewegt die aktuelle Position in der Liste auf die nächste Zeile. Diese Methode gibt `True` zurück, wenn noch weitere Zeilen folgen, ansonsten `False`.
`NextResult()`	Wenn der zugrunde liegende Befehl der Aufruf einer Stored Procedure oder eine Batch-Select-Anweisung ist, verwaltet das DataReader-Objekt mehrere Datensatzlisten. Über `NextResult` können Sie zur nächsten Datensatzliste wechseln.
`Item(`*Index*`)`	Über diese Eigenschaft erhalten Sie Zugriff auf eine Spalte der aktuellen Datenzeile. Als Index können Sie den Integer-Index der Spalte oder einen String mit dem Namen des Feldes übergeben. Die Rückgabe ist vom Typ `Object`, muss also in der Regel noch konvertiert werden.
`Close()`	Über diese Methode schließen Sie das Objekt

Tabelle 14.4: Die wichtigsten Methoden der Klassen SqlDataReader und OleDbDataReader

DataSet, DataTable, DataAdapter und DataView

Ein Objekt der Klasse `DataSet`, die im Namensraum `System.Data` definiert ist, also für alle Provider gleich gilt, repräsentiert einen Satz von Tabellen, deren Beziehungen untereinander und deren Einschränkungen (Contraints). Diese Daten werden komplett im Arbeitsspeicher des Client verwaltet und sind nach dem Einlesen von der Datenquelle getrennt. Das `DataSet`-Objekt ist damit so etwas wie das Abbild einer Datenbank (oder eines Teils einer Datenbank) im Speicher. Es erlaubt das Bearbeiten, Löschen und Anfügen von Daten.

Die `Tables`-Eigenschaft eines `DataSet`-Objekts ist eine Auflistung von `DataTable`-Objekten. Ein `DataTable`-Objekt repräsentiert eine einzelne Tabelle. Über Methoden und Eigenschaften dieses Objekts können Sie die Daten bearbeiten.

Um ein `DataSet`-Objekt und die enthaltenen `DataTable`-Objekte erzeugen zu können, benötigen Sie eine oder mehrere `SqlDataAdapter`- bzw.

OleDataAdapter-Objekte. Diese schaffen die Verbindung zwischen einem Command-Objekt und dem DataSet-Objekt und ermöglichen das Lesen, aber auch das Aktualisieren der Daten. Über die Update-Methode des DataAdapter-Objekts schreiben Sie z. B. die in den DataTable-Objekten geänderten Daten in die Datenbank.

Da das Editieren, Anfügen und Löschen von Daten in ASP.NET meist eine relativ einfache Sache ist, die kein so komplexes Objekt wie das DataSet*-Objekt benötigt, beschreibe ich dieses und die damit zusammenhängenden Klassen nicht in diesem Buch. Ein* DataSet*-Objekt macht Sinn, wenn Windowsanwendungen mit komplexen Daten arbeiten und diese Daten über eine längere Zeit im Arbeitsspeicher verwaltet werden sollen. Beim serverseitigen Verarbeiten von Daten, bei dem es eigentlich nur immer um das Einlesen der Daten und um das einfache Ausführen von SQL-Befehlen geht, reicht ein DataReader-Objekt normalerweise vollkommen aus. Sie finden nähere Informationen zu den hier nicht beschriebenen Klassen in meinem Visual Basic.NET- oder meinem C#-Buch aus der Nitty-Gritty-Reihe.*

14.2.3 Verbindung aufnehmen

SQL Server

Wenn Sie auf eine SQL Server-Datenbank zugreifen wollen, benötigen Sie zum Verbindungsaufbau ein Objekt der Klasse System.Data.SqlClient.SqlConnection. Da Sie im Quellcode öfter mit den Typen des Namensraums System.Data und System.Data.SqlClient arbeiten werden, sollten Sie diese Namensräume importieren, damit Sie die enthaltenen Typen ohne Angabe des Namensraums verwenden können. Die SqlConnection erwartet einen Verbindungsstring mit den Informationen zu der gewünschten Verbindung. Dieser String besitzt eine große Anzahl möglicher Argumente. Ich beschreibe hier nur die wichtigsten. Der Rest ist ziemlich gut in der .NET-Hilfe erläutert. Im Argument Server geben Sie den Namen des SQL Servers an, im Argument Database. Im Argument Connection Timeout können Sie den Timeout für den Verbindungsaufbau (in Sekunden) beeinflussen. Die

Voreinstellung von 15 Sekunden ist eventuell etwas kurz, wenn der Server weit entfernt ist. Schließlich definieren Sie noch im Argument `User Id` den Loginnamen des Benutzers und im Argument `Password` dessen Passwort.

Das folgende Beispiel baut eine Verbindung zu der Beispieldatenbank Pubs im lokalen SQL Server auf:

```
<%@Page Language="VB"%>
<%@Import Namespace="System.Data"%>
<%@Import Namespace="System.Data.SqlClient"%>

<html>
<head>

<title>Verbindungen</title>

<script language="VB" runat="Server">
 Private Sub Page_Load()
  ' Connection erzeugen und initialisieren
  Dim con As SqlConnection
  con = New SqlConnection("Server=(local);" & _
     "Database=Pubs;User Id=sa;Password=bandit")
  ' Verbindung öffnen
  con.Open()

  ' Erfolgsmeldung ausgeben
  lblInfo.Text = "<p>Verbindung ist geöffnet.</p>"

  ' Verbindung schließen
  con.Close()

  ' Erfolgsmeldung ausgeben
  lblInfo.Text += _
     "<p>Verbindung ist wieder geschlossen.</p>"

 End Sub
</script>
```

```
</head>

<body>

<h1>Verbindung zu einer SQL Server-Datenbank</h1>

<form Runat="server">
 <asp:Label Id="lblInfo" Runat="Server"/>
</form>

</body>
</html>
```

Natürlich müssen Sie beim Aufbau der Verbindung auf mögliche Fehler reagieren. Ich verzichte hier aber auf die Fehlerbehandlung, weil das die Beispiele zu sehr aufblähen würde. Die Fehlerbehandlung wird in Kapitel 7 behandelt.

Andere Datenquellen, für die nur ein ADO-Provider verfügbar ist

Wenn Sie eine andere Datenquelle bearbeiten wollen, für die zurzeit noch kein ADO.NET-Provider verfügbar ist, können Sie dazu den ADO.NET-Provider für ADO verwenden. Der Verbindungsaufbau zu einer Datenquelle gestaltet sich dann identisch zu ADO.

Für eine ADO-Verbindung müssen Sie ein Objekt der Klasse `OleDbConnection`. Den Verbindungsstring können Sie direkt beim Erzeugen des Objekts übergeben. Ich verwende wieder eine Datenlinkdatei (vgl. Kapitel 10), die ich deshalb im Verbindungsstring angebe. Das Beispiel baut eine Verbindung zur Nordwind-Datenbank auf. Die Datenlinkdatei befindet sich drei Ordnerebenen höher als die `aspx`-Datei:

```
<%@Page Language="VB"%>
<%@Import Namespace="System.Data"%>
<%@Import Namespace="System.Data.OleDb"%>

<html>
<head>

<title>Verbindungen</title>
```

```
<script language="VB" runat="Server">
 Private Sub Page_Load()
  ' Connection erzeugen und initialisieren
  Dim con As OleDbConnection
  con = New OleDbConnection("File Name=" & _
     Page.MapPath(".") & "\..\..\..\Nordwind.udl")
  ' Verbindung öffnen
  con.Open()

  ' Erfolgsmeldung ausgeben
  lblInfo.Text = "<p>Verbindung ist geöffnet.</p>"

  ' Verbindung schließen
  con.Close()

  ' Erfolgsmeldung ausgeben
  lblInfo.Text += _
     "<p>Verbindung ist wieder geschlossen.</p>"

 End Sub
</script>

</head>

<body>

<h1>Verbindung zu einer Datenbank über einen ADO-Provider</h1>

<form Runat="server">
 <asp:Label Id="lblInfo" Runat="Server"/>
</form>

</body>
</html>
```

14.2.4 Daten einlesen

Zum Einlesen von Daten verwenden Sie in ASP.NET idealerweise eines der DataReader-Objekte. Die Gründe dafür habe ich ja bereits genannt. Um das DataReader-Objekt erzeugen zu können, benötigen Sie zunächst ein DataCommand-Objekt. Die `ExecuteReader`-Methode dieses Objekts gibt ein DataReader-Objekt zurück:

```
' Connection erzeugen und initialisieren
Dim con As OleDbConnection
con = New OleDbConnection("File Name=" & _
   Page.MapPath(".") & "\..\..\..\Nordwind.udl")
' Verbindung öffnen
con.Open()
' Command-Objekt zum Einlesen der Daten erzeugen
' Die Connection wird im zweiten Argument übergeben
Dim cmd As OleDbCommand = New _
  OleDbCommand("SELECT * FROM Artikel", con)

' DataReader über die ExecuteReader-Methode
' des Command-Objekts erzeugen
Dim dr As OleDbDataReader
dr = cmd.ExecuteReader()
```

Zur Anzeige der Daten binden Sie das DataReader-Objekt normalerweise einfach an ein `DataGrid`-, `DataList`- oder `Repeater`-Objekt an. Wie das geht, habe ich ja bereits beschrieben. Ich zeige hier aber trotzdem, wie Sie die Daten »von Hand« einlesen:

```
' Sequenziell durch die gelesenen Daten gehen
Do While dr.Read()
   ' Spaltenwerte auslesen
   Response.Write(dr.Item("Artikelname") & "<br>")
Loop
```

Die `Read`-Methode stellt den Zeilenzeiger auf den nächsten Datensatz. Nach dem Öffnen steht dieser Zeiger (anders als bei ADO) zunächst vor dem ersten Datensatz. Deshalb führt der erste `Read`-Aufruf dazu, dass der Zeiger auf den ersten Datensatz gestellt wird. `Read` gibt `True` zurück, wenn noch Datensätze folgen. Die Schleife läuft also so lange, bis keine Datensätze mehr vorhanden sind.

Nach dem Lesen sollten Sie das Objekt noch schließen:

```
' DataReader schließen
dr.Close()
```

14.2.5 Daten editieren, anfügen und löschen

Das Editieren, Anfügen und Löschen von Daten programmieren Sie unter ADO.NET, wie auch bei ADO, idealerweise über SQL-Anweisungen. Der Aufwand, der notwendig ist, wenn Sie für das Bearbeiten einer Datenquelle ein `DataSet`-Objekt verwenden, ist einfach zu groß. Die SQL-Anweisung führen Sie dann über die Methode `ExecuteNonQuery` eines Objekts der Klasse `SqlCommand` bzw. `OleDbCommand` aus.

Wenn Sie mich in die Riege der Autoren der Pubs-Datenbank aufnehmen wollen, weil Sie dieses Buch nun endlich bis zum Ende durchgelesen haben, verwenden Sie den folgenden Quellcode:

```
Dim con As SqlConnection = New SqlConnection()
con.ConnectionString = "Server=(local);Database=Pubs;" & _
   "User Id=sa;Password=bandit"
con.Open()
Dim cmd As SqlCommand = New SqlCommand( _
   "INSERT INTO Authors (au_id, au_lname, " & _
   "au_fname, contract) VALUES " & _
   "('333-42-3333', 'Bayer', 'Jürgen', 0)", con)
cmd.ExecuteNonQuery()

con.Close()
```

Und zuletzt:

```
Book.Close()
```

A Anhang

A.1 Literaturverzeichnis

Anderson, Richard u. A.: Professional ASP.NET. Wrox Press Ltd. Birmingham 2001.

Coolnerds (Hrsg.): Coolnerds Electronic JavaScript Reference. *http://www.coolnerds.com/jscript/javaref.htm*

Dellwig Ingo: Nitty-Gritty HTML. Addison Wesley. München 2000.

Geese, Elmar; Heiliger, Markus: XML mit VB und ASP. Galileo Computing. Bonn 2000.

Microsoft (Hrsg.): DHTML References. *http://msdn.microsoft.com/workshop/author/dhtml/reference/dhtmlrefs.asp*.

Münz, Stefan. SelfHTML. HTML-Dateien selbst erstellen. *http://www.teamone.de/selfhtml/.*

Netscape (Hrsg.): Client-Side JavaScript Reference. *http://developer.netscape.com/docs/manuals/js/client/jsref/contents.htm*.

W3C (Hrsg.): Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. *http://www.w3.org/TR/2000/REC-xml-20001006.*

W3C (Hrsg.): HTML 4.01 Specification. W3C Recommendation 24 December 1999. *http://www.w3.org/TR/html401/*.

W3C (Hrsg.): XHTML™ 1.1 - Module-based XHTML. W3C Recommendation 31 May 2001. *http://www.w3.org/TR/2001/REC-xhtml11-20010531/*.

Web Design Group (Hrsg.): HTML 4.0 Reference. *http://www.htmlhelp.com/reference/html40/*.

Webreview (Hrsg.): Overview of the CSS Specification. http://www.webreview.com/style/css1/glossary.shtml.

Wium Lie, Håkon, Bos, Bert: Cascading Style Sheets. Level 1. W3C Recommendation 17 Dec 1996, revised 11 Jan 1999. *http://www.w3.org/tr/rec-css1*.

Stichwortverzeichnis

Symbole

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z